National Instruments

Model GPIB-MAC User Manual

Part Number 320064-01

November 1985 Edition

National Instruments
12109 Technology Boulevard
Austin, Texas 78727
(512) 250-9119

LIMITED WARRANTY

The GPIB-MAC is warranted against defects in materials and workmanship for a period of one year from date of shipment. National Instruments will repair or replace equipment which proves to be defective during the warranty period. This warranty includes parts and labor. A Returned Material Authorization (RMA) number must be obtained from the factory before any equipment is returned for warranty. During the warranty period, the owner may return failed parts to National Instruments for repair. National Instruments will pay the shipping costs of returning the part to the owner. All items returned to National Instruments for repair must be clearly marked on the outside of the package with a RMA. No other warranty is expressed or implied National Instruments shall not be liable for consequential damages. Contact National Instruments for more information.

IMPORTANT NOTE

The material in this manual is subject to change without notice. National Instruments assumes no responsibility for errors which may appear in this manual. National Instruments makes no commitment to update, nor to keep current, the information contained in this document.

Trademarks

FCC RADIO FREQUENCY INTERFERENCE COMPLIANCE

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested using a shielded serial I/O cable and standard GPIB cable and found to comply within the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment Operation of this equipment in a residential area is likely to cause interference, in which case the user at his own expense will be required to take whatever measures may be necessary will be required to take whatever measures may be necessary to correct the interference.

If the equipment does cause interference to radio or television reception, which can be determined by turning the equipment on and off, one or more of the following suggestions may reduce or eliminate the problem.

* Operate the equipment and the receiver on different branches of your AC electrical system.

* Move the equipment away from the receiver with which it is interfering.

* Reposition the equipment or receiver.

* Reposition the receiver's antenna.

* Unplug any unused I/O cables. Unterminated I/O cables are a potential source of interference.

* Remove any unused circuit boards. Unterminated circuit boards are also a potential source of interference.

* Be sure the computing device is plugged into a grounded outlet and that the grounding has not been defeated with a cheater plug.

* Replace the GPIB cable with Hewlett-Packard Model 10833 cable.

If none of these measures resolves your interference problems, contact the manufacturer or write to the U.S. Government Printing Office, Washington, D.C. 20402, for the booklet, “How to Identify and Resolve Radio-TV Interference Problems, “Stock Number 004-000-000345-4.

Preface

Welcome to the family of National Instruments GPIB products and to the Model GPIB-MAC.

The Model GPIB-MAC allows the GPIB to be controlled from a Macintosh personal computer.

About the Manual

The manual is designed for users who have some familiarity with the Macintosh personal computer, the GPIB, and test and measurement equipment.

For users with less experience, we have included appendixes that describe the operation of the GPIB.

For more specific details on the operation of the GPIB, refer to the IEEE Std. 488-1978, "IEEE Standard Digital Interface for Programmable Instrumentation."

Whatever your level of experience, if you encounter problems, National Instruments has a staff of applications engineers ready to help you with your particular problem. Just call

800/53 l-GPIB
800/531-5066 (outside Texas)
800/IEEE-488 (inside Texas)

between the hours of 8:00 a.m and 5:00 p.m, Central Time.

Now, look over the next few pages at how the manual is organized and then at the Table of Contents so that you will be familiar with the complete contents for future reference.

Organization of the Manual

Section One - gives brief introductions to the GPIB-MAC and the IEEE-488.

Section Two - contains the installation and configuration steps.

Section Three - explains how to program the GPIB-MAC.

Section Four - gives a detailed description of each function. The function descriptions are arranged in alphabetical order and each contains the syntax and purpose of the functions, and examples.

Appendix **A** - contains a table of multiline interface messages.

Appendix B - lists status information.

Appendix C - shows how to change the operating voltage from 115 to 230.

Appendix D - describes the operation of the GPIB.

Appendix F - gives answers to common questions.

Appendix F - explains the use and operation of Parallel Polls.

Appendix G - gives additional detail on setting switches on the GPIB-MAC.

Appendix H - contains a sample program of general programming steps.

Appendix1 - contains a sample program to control the serial port of the Macintosh from a C program.

TABLE OF CONTENTS

* frequently used function

LIST OF FIGURES

LIST OF TABLES

Section One - Introduction

This section provides brief introductions to the GPIB-MAC and the IEEE-488. It also describes the physical, electrical, and environmental characteristics of the GPIB-MAC.

Introduction to the GPIB-MAC

The GPIB-MAC is a high performance GPIB-to-Macintosh interface. The GPIB-MAC together with a Macintosh personal computer provide a means of Controlling, Talking, and Listening on the GPIB.

The GPIB-MAC has all the software and logic required to implement the physical and electrical specifications of the IEEE-488. It is capable of interpreting and executing high level commands that you send to it over the Macintosh serial port.

Introduction to the IEEE-488 (GPIB)

The IEEE-488, also known as the General Purpose Interface Bus or GPIB, is a high speed parallel bus structure originally designed by Hewlett-Packard. It is generally used to connect and control programmable instruments, but has gained popularity in other applications, such as intercomputer communication and peripheral control.

The specifications of the GPIB are too lengthy and comprehensive to be explained in this manual. However, Appendix D, 'Operation of the GPIB," contains a summary of pertinent IEEE-488 information you might find useful.

Description of the Model GPIB-MAC

Included here are the GPIB-MAC environmental, physical, and electrical specifications.

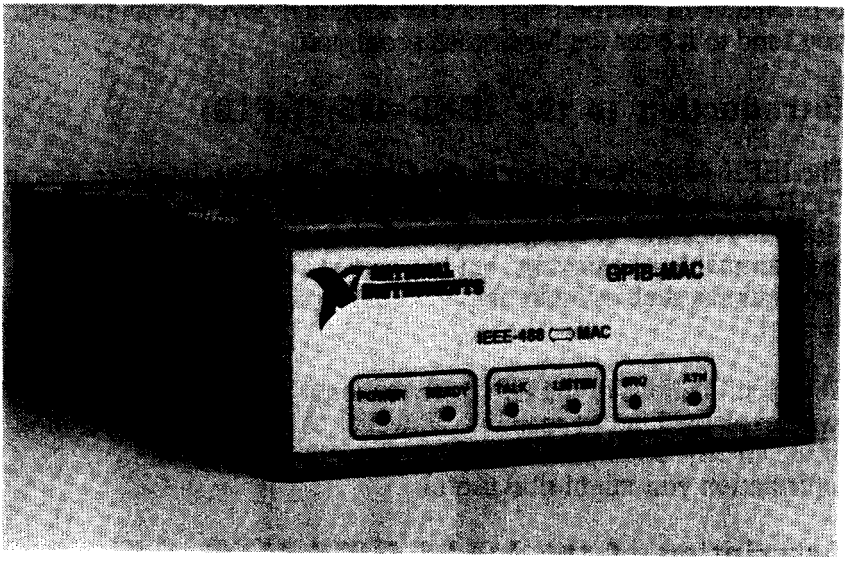
Environmental Specifications

The GPIB-MAC is designed to operate in temperatures ranging from 10 to 40 degrees Celsius, and in humidity ranging from 10% to 95% non-condensing.

The GPIB-MAC can be stored in temperatures ranging from 0 to 70 degrees Celsius.

Physical Specifications

The GPIB-MAC, shown in the following figure, is housed in a structural foam injection molded case. The unit can be rack mounted or placed on a table.



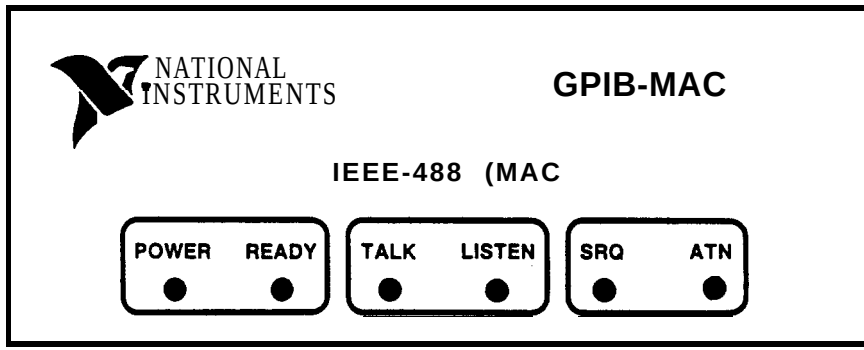
Model GPIB-MAC

Quick Reference Chart.

On the back panel of the GPIB-MAC is a reference chart that contains the information you need to configure the rear panel switches of the GPIB-MAC. More detailed configuration information is included in Section Two.

The Model GPIB-MAC Front Panel

The front panel of the GPIB-MAC is shown in the following figure. The six light emitting diodes (LEDs) show the current status of the GPIB-MAC.



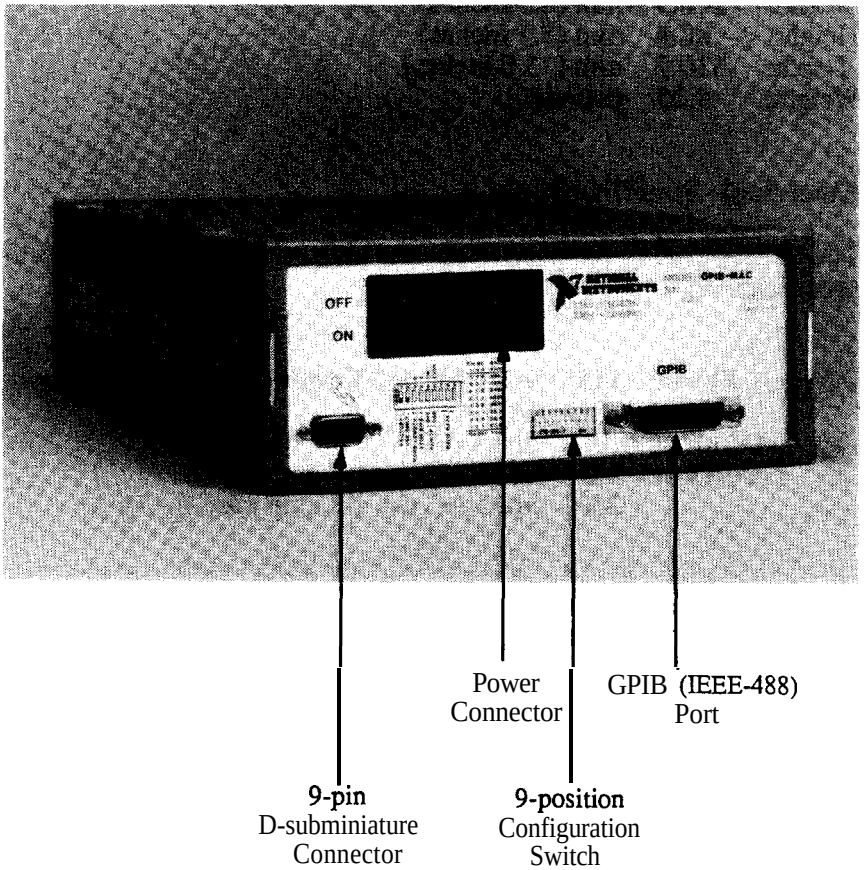
Front Panel of the GPIB-MAC

The following table shows what each LED indicates when lit.

LED	Purpose
POWER	indicates power is on.
READY	indicates that the power-on self-test has passed successfully and unit is ready to operate.
TALK	indicates that the GPIB-MAC is currently addressed to Talk on the GPIB.
LISTEN	indicates that the GPIB-MAC is currently addressed to Listen on the GPIB.
ATN	indicates that the GPIB signal line ATN* is asserted (low).
SRQ	indicates that the GPIB signal line SRQ* is asserted (low).

The Model GPIB-MAC Back Panel.

The back panel of the GPIB-MAC is shown in the following figure. The power connector, 9-position configuration switch, 9-pin D-subminiature connector, and GPIB (IEEE-488) port are shown.



Back Panel of the Model GPIB-MAC

Mechanical Specifications

Width:	216.9 mm (8.5 inches)
Height:	88.4 mm (3.5 inches)
Depth:	330.2 mm (13.0 inches)
Weight:	5.25 pounds

Electrical Specifications

The GPIB-MAC is designed to operate under the following electrical specifications.

Power:	115	volts AC or 230 volts AC; 50/60 Hz; 20 VA
Typical Current:	.09	amps AC
Fuse Type:	115	volts AC use 1/4 amp Fast
	230	volts AC use 1/8 amp Fast

Section Two - Installation and Configuration

Use this section to install and configure the GPIB-MAC. Then, read Sections Three and Four to learn about how to program the GPIB-MAC.

Inspection

Before you install the GPIB-MAC, inspect the shipping container and its contents for damage. If damage appears to have been caused in shipment, file a claim with the carrier. Retain the packaging material for possible inspection or for reshipment.

If the equipment appears to be damaged, do not attempt to operate it. Contact National Instruments for instructions.

Installation

There are four basic steps to installing the GPIB-MAC.

1. Verify voltage requirements
2. Configure GPIB-MAC rear panel switches
3. Connect cables
4. Turn power switch to On

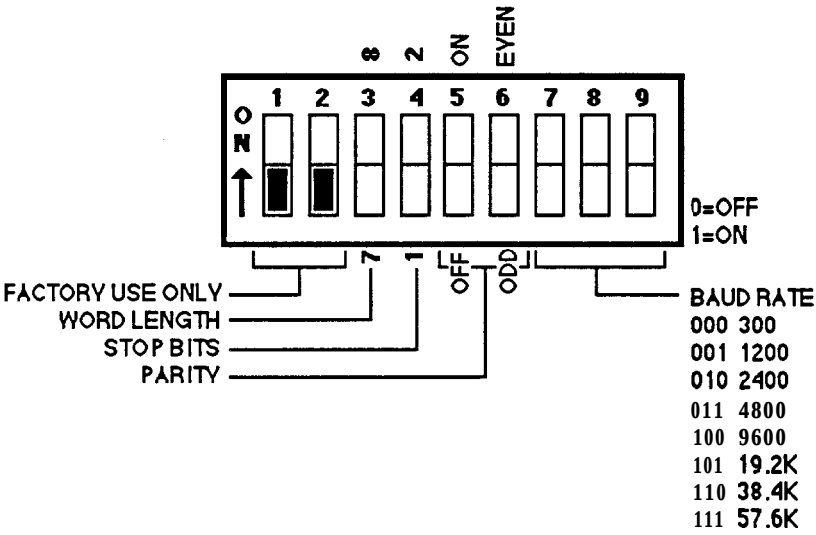
Voltage Requirements

The GPIB-MAC is shipped with the internal voltage selector switch configured to operate on a standard 115 VAC power line.

If your setup requires 230 VAC, refer to Appendix C to learn how to open the box to change the voltage selector switch.

Configure the GPIB-MAC Rear Panel Switches

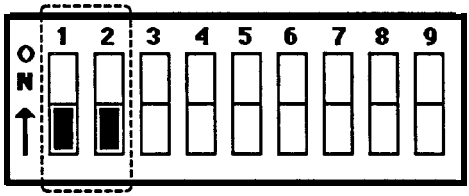
Configure the GPIB-MAC by setting the switches of the 9-pin, rear panel DIP switch. The following figure shows the switches labeled for factory use only, 7- or 8-bit word length, one or two stop bits, odd/even or no parity, and baud rate.



GPIB-MAC DIP Switch

Switches 1 and 2 - Factory Use Only

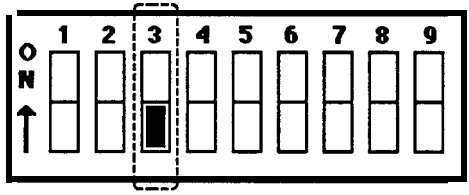
Switch 1 and 2 are for factory use only and should always be positioned in the off position as shown below.



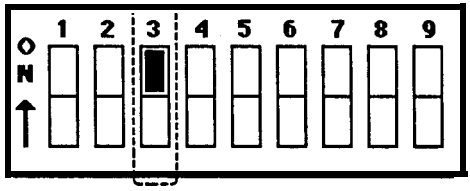
Factory Use Only

Switch 3 - Word Length

Configure the GPIB-MAC for a serial word length of 7 or 8 bits by setting switch 3. The off position indicates 7 bits, the on position indicates 8.



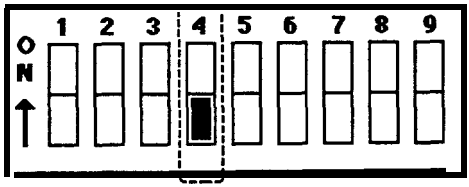
7-bit Word Length



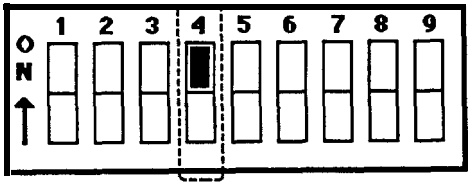
8-bit Word Length

Switch 4 - Stop Bits

Configure the GPIB-MAC for a stop bit length of 1 or 2 bits by setting switch 4. The off position indicates 1 bit, the on position indicates 2.



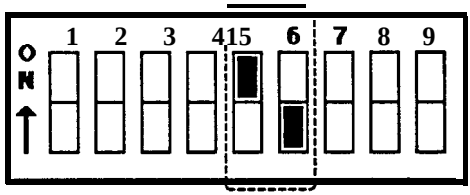
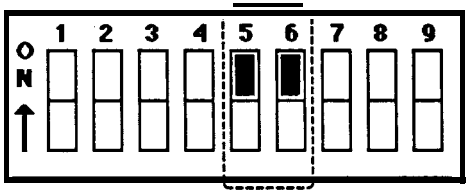
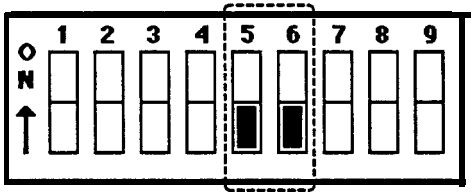
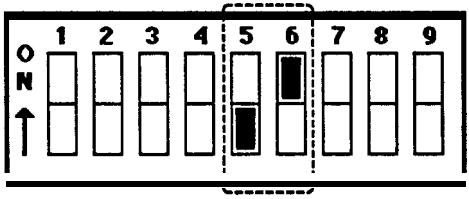
1 Stop Bit



2 Stop Bits

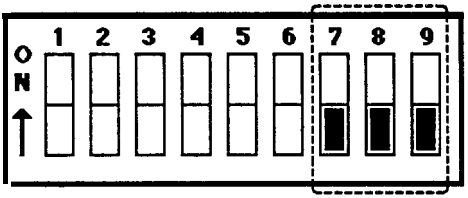
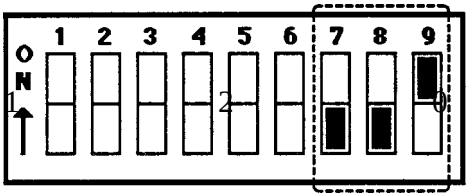
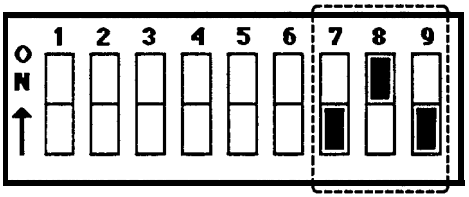
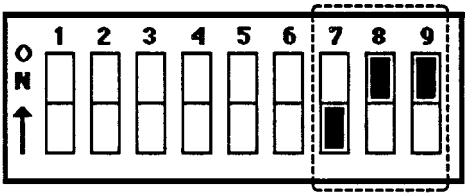
Switches 5 and 6 - Parity Type

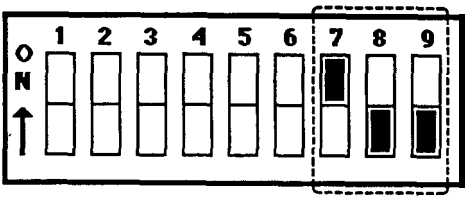
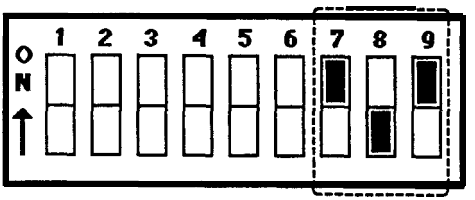
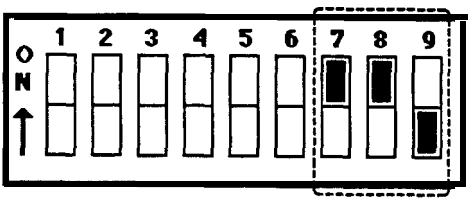
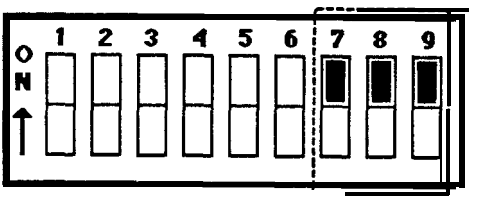
The GPIB-MAC can transmit and receive serial data using odd parity, even parity, or no parity. Configure the GPIB-MAC for the correct parity according to the following figure. Note that switch 5 indicates parity off. Switch 6 indicates parity odd or even. If switch 5 is set to off, switch 6 is ignored.

Switch Settings	Parity Type
	Odd Parity
	Even Parity
	Parity Inhibit
	Parity Inhibit

Switches 7, 8, and 9 - Baud Rate

Configure the GPIB-MAC for the appropriate baud rate by setting switches 7, 8, and 9 according to the following figures.

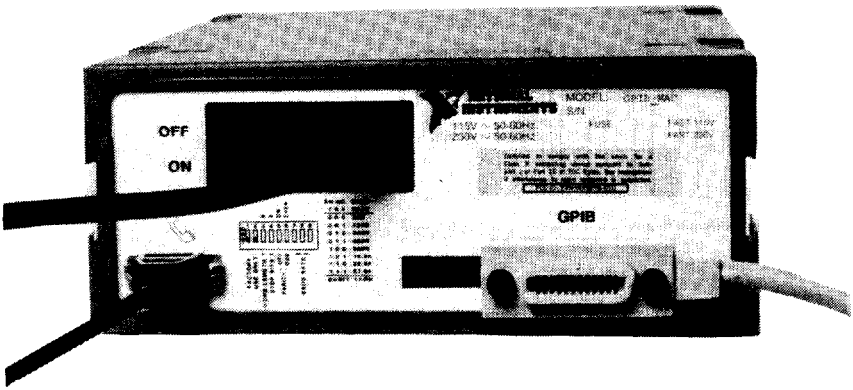
Switch Settings	Baud Rate
	300
	0
	2400
	4800

Switch Settings	Baud
	9600
	19.2K
	38.4K
	57.6K

If you need more information on how the switches of the GPIB-MAC should be set, refer to Appendix G.

Connect Cables

You must connect three cables to operate the GPIB-MAC: the serial cable, the power cable, and the GPIB cable. The three cables all connect to the GPIB-MAC via the rear panel, as shown in the following figure.



GPIB-MAC with Serial Cable, Power Cable, and GPIB Cable

The power cord receptacle is located at the top left of the rear panel. If 115 VAC is required, use the standard power cable supplied with the unit. If 230 VAC is required, use a cable that is compatible with both the GPIB-MAC power receptacle and the 230 VAC plug configuration.

The GPIB cable should be connected to the 24-pin GPIB connector on the lower right of the rear panel. The connectors can be piggy-backed to add more GPIB devices to the bus. Connect the serial cable to the 9-pin, D-subminiature connector on the back of the GPIB-MAC. Connect the other end of the cable to your Macintosh modem port.

Turn Power Switch to On

The power switch is located at the left of the power cord receptacle. Turn the switch to On. There will be a slight delay while the unit performs a self-test of all the internal circuitry. The test does not affect devices connected to the GPIB-MAC. When the test successfully completes, the BEADY LED comes on. If it does not, verify your switch settings and the power connections. If the BEADY LED still fails to come on, contact National Instruments.

Note - The rear panel switches are read by the fiiware only when the unit is powered on. After changing switch settings, turn the power switch off and back on for the change to take effect. Also, the GPIB address of the GPIB-MAC at power on is 0 with secondary addressing disabled. You can change these values after power-on using a programming message.

Section Three - Programming the GPIB-MAC

This section shows how to program the GPIB-MAC by using programming messages and data strings. It describes programming messages, their format, and how they areprocessed, along with the functions and function arguments that make up the programming messages.

Programming Messages

You program the GPIB-MAC by sending it programming messages (which are ASCII strings) and data strings by way of its serial port.

Programming Message Format

A programming message consists of a function name, one or more arguments (optional), followed by a carriage return (<CR>), a linefeed (<LF>), or a carriage return followed by a linefeed (<CR><LF>).

You may enter programming messages in any combination of uppercase and lowercase letters.

Example of a Programming Message

The following line of BASIC code:

```
PRINT #1,"clr 3,4"
```

contains the function name **clr** and the arguments 3 and 4. This programming message tells the GPIB-MAC to clear the devices at GPIB addresses 3 and 4. PRINT #1 is the BASIC command to send characters to the serial port after the serial port has been opened with the "OPEN COM..." statement. In this example, BASIC automatically sends a <CR>, so it is not necessary to include it here.

The cmd and **wrt** programming messages are followed by a data string which may contain 7- or 8-bit data

Example of a Programming Message with Data String

The following lines of BASIC code:

```
PRINT # 1, "wrt 2"
PRINT #1, "IN;CI;"
```

contain the function name **wrt, the** argument 2, and the data string "IN;CI;". This programming message is telling the GPIB-MAC to write to the device at primary address 2.

"IN;CI;" is the data string which contains the data **wrt** will send out on the GPIB. In this case, a <CR> is automatically sent by BASIC following each print string, so, again, it is not necessary to include it here.

How Messages are Processed

The GPIB-MAC processes a programming message on a line-by-line basis. The GPIB-MAC buffers the entire message, interprets the function name and arguments, then executes the message.

The data portions of the **wrt** and **cmd** functions are not processed on a line-by-line basis. The data immediately following a **wrt** and a **cmd are** sent directly to the GPIB.

Function Names

The function names have been selected to indicate each function's purpose, thereby making your programs easy to understand. However, if you wish to reduce some overhead in your program and do not mind giving up these advantages, you may use only as much of the function name as is necessary to distinguish it from other functions. This abbreviated form of the function name is shown in **boldface in the** function tables and in the syntax portions of the function descriptions.

For example, the **wait** function may be called using either of the next two statements:

```
PRINT #1, "wait \x5000"
PRINT #1, "wa \x5000"
```

Function Argument Delimiters

When you type in a function, separate the first argument from the function name with at least one space. Separate each additional argument with at least one space or a comma.

In the syntax portions of the function descriptions in Section Four the square brackets ([]) are optional. If you want to include optional information, you do not need to type the brackets, only the information inside the brackets.

Abbreviations for Arguments

The function descriptions in Section Four use abbreviations for some arguments. They are as follows:

addr a GPIB address
alist one or more addrs
bool a boolean value: 1 = true, on, or enable
0 = false, off, or disable

GPIB Address

Each device on the GPIB has a GPIB address. The GPIB-MAC's address is 0 at power on and may be changed using the **caddr** function. Refer to the manuals of your GPIB devices to learn their addresses. You will need to know these when you begin to program the GPIB-MAC.

Only the lower five bits of each GPIB address are significant. These bits may be in the range from 0 through 30 for both the primary and the secondary address. For example, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. A plus sign (+) separates the primary address from the secondary address. The listen address is 32 (primary address plus 32), the talk address is 64 (primary address plus 64), and the secondary address is 2 or 98, which are equivalent. The next paragraph explains the \x notation.

0+2 or 0+98 or 32+98 or 0+\x62

Numeric String Argument

Another type of argument is a numeric string. A numeric string represents an integer, which you may express using decimal, octal, or hexadecimal digits. To specify an octal integer, precede it with a backslash (\). To specify a hexadecimal integer, precede it with a backslash x (\x) or backslash X (\X).

Each of the following numeric strings represents the decimal integer value 112:

112
\160
\x70

The GPIB address argument described previously under "GPIB Address" consisted of one or two numeric strings.

Status Information

The function descriptions in Section Four explain that the GPIB-MAC "records" specific status and error information. This means that it stores that information in its memory so that the status information is available to you when you request it.

The function descriptions also explain that the GPIB-MAC "returns to you" certain information. This means that the GPIB-MAC sends information to you over the serial port. You then read this information from your serial port

Serial Port Error Handling

The GPIB-MAC continuously monitors the serial port for transmission errors. If it encounters an error in the serial data, the GPIB-MAC records the error. You can program the GPIB-MAC to stop processing the programming message when a serial port error occurs or to ignore these serial port errors. Refer to the **spign** function.

GPIB Read and Write Termination Method (END and EOS)

You program the GPIB-MAC to Talk in order to send data messages over the GPIB, and to Listen in order to receive data messages from the GPIB.

The IEEE-488 specification defines two ways that GPIB Talkers and Listeners may identify the last byte of data messages: END and EOS. The two methods permit a Talker to send data messages of any length without the Listener(s) knowing in advance the number of bytes in the transmission.

* END message — the Talker asserts the EOI (End or Identify) signal while the last data byte is being transmitted. The Listener stops reading when it detects a data byte accompanied by EOI.

* EOS character — the Talker sends an EOS (end-of-string) character at the end of its data suing. The Listener stops receiving data when it detects the EOS character. Either a 'I-bit ASCII character or a full 8-bit binary byte may be used.

The two methods can be used individually or in combination. It is important that the Listener be configured to detect the end of a transmission.

The GPIB-MAC always terminates GPIB **rd** operations on the END message. Using the eos and eot functions, you may change the other default GPIB read and write termination methods.

Default Settings

The following tables list power-on characteristics of the GPIB-MAC and the functions you can use to change those characteristics.

SERIAL PORT CHARACTERISTICS

Characteristic	Power-on Value	Function
echo bytes to serial port	no	echo
ignore serial port errors	yes	spign
send XON/XOFF	no	xon
recognize XON/XOFF	no	xon

GPIB CHARACTERISTICS

Characteristic	power-on Value	Function
primary/secondary address	pad=0,sad=none	caddr
end-of-string modes	none	eos
send END on writes	ye	eot
ist bit setting	0	ist
GPIB-MAC is System Controller	yes	rsc
I/O timeout	10 sec	tmo
serial poll timeout	.1 sec	tmo

List of Functions by Group

The GPIB-MAC functions are divided into three main groups: GPIB functions, Serial Port functions, and General Use functions.

GPIB Functions

The GPIB functions are divided into subgroups as shown. The subgroups are listed with the most frequently used groups first. Often, the I/O and bus management functions are the only ones you need.

I/O FUNCTIONS

Function	Purpose
RD count,address	Read data
WRT count,address list data	write data

BUS MANAGEMENT FUNCTIONS

Function	Purpose
CLR address list	Clear specified device(s)
LOC address list	Go to Local
TRG address list	Trigger selected device(s)

GPIB INITIALIZATION FUNCTIONS

Function	Purpose
CADDR address	Change the GPIB address of the GPIB-MAC
EOS modes,eoschar	Change/disable GPIB EOS termination mode
EOT on/off	Enable/disable END message on GPIB writes
ONL on/off	Place the GPIB-MAC online/ offline
RSC on/off	Request or release System Control
TMO values	Change or disable time limit

SERIAL POLL FUNCTIONS

Function	Purpose
RSP address list	Request (conduct) a serial poll
RSV status byte	Request service and/or set or change the serial poll status byte

LOW-LEVEL CONTROLLER FUNCTIONS

Function	Purpose
CAC mode	Become active controller
CMD count commands	Send IEEE-488 commands
GTS mode	Go from Active Controller to Standby
PCT address	Pass Control
SIC time	Send interface clear
SRE on/off	Set/clear remote enable

PARALLEL POLL FUNCTIONS

Function	Purpose
IST on/off	Set or clear individual status bit for use in GPIB-MAC response to Parallel Polls
PPC values	Parallel Poll Configure
PPU address list	Parallel Poll Unconfigure
RPP	Request (conduct) a Parallel Poll

Serial Port Functions

SERIAL PORT INITIALIZATION FUNCTIONS

Function	Purpose
ECHO on/off	Echo characters received from serial port
SPIGN on/off	Ignore serial port errors
XON modes	Change serial port XON/XOFF protocol

General Use Functions

GENERAL USE FUNCTIONS

Function	Purpose
IDMAC	Identify system
STAT modes	Return GPIB-MAC status
WAIT mask	Wait for selected event

List of Functions in Alphabetical Order

The following is an alphabetical list of all functions.

GPIB-MAC FUNCTIONS

Function	Purpose
CAC mode	Become active controller
CADDR address	Change GPIB address of the GPIB-MAC
CLR address list	Clear specified device(s)
CMD count commands	Send GPIB commands
ECHO on/off	Echo characters received from serial port

GPIB-MAC FUNCTIONS (CONTINUED)

EOS modes,eos	Change/disable GPIB EOS termination mode
EOT on/off	Enable/disable END message on GPIB writes
GTS mode	Go from Active Controller to Standby
IDMAC	Identify system
IST set/clear	Set or clear individual status bit for use in GPIB-MAC response to Parallel Polls
LOC address list	Go to Local
ONL on/off	Place the GPIB-MAC online/offline
PCT address	Pass Control
PPC values	Parallel Poll Configure
PPU address list	Parallel Poll Unconfigure
RD count,address	Read data
RPP	Request (conduct) a Parallel Poll
RSC on/off	Request or release System Control
RSP address list	Request (conduct) a serial poll
RSV serial poll	Request service/set or change the serial poll status byte
SIC time	Send interface clear
SPIGN on/off	Ignore serial port errors
SRE on/off	Set or clear remote enable
STAT modes	Return GPIB-MAC status
TMO values	Change or disable time limit
TRG address list	Trigger selected device(s)
WAIT mask	Wait for selected event
WRT count,address list data	write data
XON modes	Change serial port XON/XOFF protocol

Section Four - Functions

This section contains descriptions of functions which you use to program the GPIB-MAC. These functions are in alphabetical order and are formatted to provide you a handy reference.

Points to Remember

1. The programming examples for each function description are in Microsoft BASIC version 2.0.

2. In the syntax portion of the function descriptions, arguments enclosed in brackets are optional. Do not enter the brackets as part of your argument.

3. Terminate each programming message with a carriage return (<CR>), a linefeed (<LF>), or a carriage return followed by a linefeed (<CR><LF>). The terminator is denoted by <CR> in the syntax portions of the function descriptions. In the programming examples, the BASIC PRINT # statement automatically sends a carriage return at the end of the string, so a carriage return is not placed there explicitly.

4. To send more than one programming message per PRINT statement, embed a <CR> (denoted by CHR$(13)) or <LF> (denoted by CHR$(10))in the statement. For example, to send the two programming messages "send interface clear" and "send remote enable," you could use either of these two sequences:

```
PRINT #1,"sic"
PRINT #1,"sre 1"

    or

PRINT # 1 ,"sic"+CHR$(13)+"sre 1"
```

5. For all examples, the communications port has been assigned to file number 1 (#1) by the BASIC OPEN "COM..." statement.

6. It is necessary for you to send only enough characters of the function name to distinguish it from other functions. Those characters are shown in boldface in the syntax portion of each function description.

7. I/O and bus management functions meet most of your needs. In the descriptions that follow, these frequently used functions are marked with an asterisk (*).

cac - Become Active Controller

cac: Low-level Controller function

Syntax: **cac** [bool]<CR>

Purpose: You use **cac** to change the GPIB-MAC from Standby Controller to Active Controller and when the I/O and bus management functions do not meet the needs of your device. **cac** allows you more precise control over the GPIB than the I/O and bus management functions.

Remarks: If the argument **bool** is 1, the GPIB-MAC takes control immediately; that is, it takes control asynchronously. If the argument **bool** is 0, the GPIB-MAC takes control after any handshake that is in progress completes; that is, it takes control synchronously.

If you call **cac** without an argument, the GPIB-MAC returns to you the current controller status, which is 0 if the GPIB-MAC is not the Active Controller and1 if the GPIB-MAC is the Active Controller.

If call **cac** with an argument and the GPIB-MAC is not CIC, the GPIB-MAC records the ECIC error.

The power-on Controller status of the GPIB-MAC is Idle Controller.

Refer also to **gts** and **sic.**

Examples:

1. PRINT #1,"cac 1" 'Take control immediately.

2. PRINT # 1 ,"cac 0" 'Take control synchronously.

3. PRINT #1,"CAC" 'Are we the active controller?

response: 1<CR><LF> ...yes...we're CAC

caddr - Change GPIB Address of the GPIB-MAC

caddr: Initialization function

Syntax: **caddr** [addr]<CR>

Purpose: You use **caddr** at the beginning of your program to change the GPIB address of the GPIB-MAC.

Remarks: The argument **addr** is a device address that specifies the new GPIB address for the GPIB-MAC. **addr** consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+). Both addresses are expressed as numeric strings.

Only the lower five bits of each address are significant. These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

If you specify a primary address without a secondary address, secondary addressing is disabled.

If you call **caddr** without an argument, the GPIB-MAC returns to you its current GPIB address.

The address assigned by this function remains in effect until you call **caddr** again, call **onl,** or you turn off the GPIB -MAC.

The power-on default is 0 with secondary addressing disabled.

Examples:

1. PRINT #1,"caddr 0+22" ‘Give GPIB-MAC a primary address ‘of 0 and a secondary address of 22.

2. PRINT #1,"CADDR 1” ‘Change GPIB-MAC primary ‘address to 1 and disable secondary ‘addressing.

3. PRINT #1,"CADDR" ‘Return current GPIB-MAC address.

response: 1<CR><LF>

clr - Clear. Specified Device *

clr: **Bus** Management function

Syntax: clr [alist]<CR>

Purpose: You use **clr** to reset the internal or device functions of the specified devices. For example, a multimeter might require that you send it either the GPIB Device Clear or Selected Device Clear command to change its function, range, and trigger mode back to its default setting. Use **clr** to do this.

Remarks: The argument **alist** is a list of **addrs** separated by commas or spaces. **addrs are** device addresses that specify the GPIB addresses you wish to clear. **alist** may consist of only one **addr.**

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

If you call **clr** with **alist, the GPIB-MAC** clears only the devices specified in **alist** (Selected Device Clear).

If you call **clr** without **alist, the GPIB-MAC** clears all devices (Device Clear).

If this is the first function you call that requires GPIB controller capability, and you have not disabled System Controller capability with csc, the GPIB-MAC sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

Refer to Appendix D for more information on clearing devices.

Refer to Appendix B for more error information.

Examples:

1. PRINT #1,"clr 14+30,16+12,18,3+26,6" 'Selectively clear '5 devices.

2. PRINT #1,"CLR" 'Issue Device Clear 'to all devices.

* frequently used function

cmd - Send GPIB Commands

cmd: Specialized Controller function

Syntax: cmd [#count]<CR>
commands<CR>

Purpose: **You** use **cmd** when the I./O and bus management functions do not meet the needs of your device. **cmd** allows you precise control over the GPIB. For example, in applications that require command sequences not sent by other functions, **cmd allows you** to transmit any sequence of interface messages (commands) over the GPIB .

Remarks: The argument **count** is a numeric string preceded by a number sign (#). **#count** specifies the number of GPIB command bytes (interface messages) to send, which is a number between 1 and 255. The number of command bytes must not include the carriage return or linefeed that you include to indicate the end of the programming message.

The argument **commands** is a list of GPIB commands. These commands are represented by their ASCII character equivalents. For example, the GPIB Untalk (UNT) command is the ASCII character underscore (_).

Refer to Example 2 to learn how to send non-printable characters.

If you call **cmd** without **#count, the** GPIB-MAC recognizes the end of the command string when it sees a <CR> or <LF>. **#count** is required only if the command string contains a <CR> or an <LF> character. However, a <CR> or an <LF> in the command string would be unusual since neither of these is a defmed GPIB command.

The GPIB commands, or interface messages, are listed in Appendix A. They include device talk and listen addresses, secondary addresses, messages, device clear and trigger instructions, and other management messages.

Do not use **cmd** to send programming instructions to devices. Use **rd** and **wrt** to send or receive device programming instructions and other device dependent information.

The cmd operation terminates when:

- the GPIB-MAC successfully transfers all commands,
- the GPIB-MAC detects an error (GPIB-MAC is not CIC),
- the **I/O** time limit is exceeded,
- the Take Control (TCT) command is in your command string and is sent to the GPIB.
- the Interface Clear (IFC) message is received from System Controller (not the GPIB-MAC).

After **cmd** terminates, the GPIB-MAC records the number of command bytes it actually sent. If an event in the above list occurs, the count may be less than expected

If you specify **#count** and enter more than **#count** command bytes, the excess command bytes up to the <CR><LF> are discarded.

If you call **cmd** and the GPIB-MAC is not CIC, the GPIB-MAC records the ECIC error.

If the GPIB-MAC is CIC but not Active Controller, it takes control and asserts ATN before sending the command bytes. It remains Active Controller afterward.

Refer to Appendix A to convert hex values to ASCII characters.

Examples:

1. PRINT #1,"CMD" 'Program device at address 11 to
'listen and GPIB-MAC at address 0
'to talk.

PRINT #1,"+@" 'Device listen address is 43 or
'ASCII + and GPIB-MAC talk
'address is 64 or ASCII @.

PRINT # 1,' 'WRT" Write the string "ABCDE" to
PRINT # 1 ," ABCDE" 'device at address 11.

2. PRINT #1,"cmd"+CHR$(13)+" ?W"+CHR$(9)
'Pass control to device 23
'(CHR$(9)=TCT command).

echo - Echo Characters Received from Serial Port

echo: Serial Port function

Syntax: **echo** [bool]<CR>

Purpose: You use **echo** when a terminal emulation program is run on the Macintosh while connected to the GPIB-MAC and you wish to echo what you type on the screen.

Remarks: If the argument **bool** is 1, characters received from the serial port are echoed back to the serial port If the argument **bool** is 0, characters are not echoed. If the argument **bool** is 1 and echoing was previously disabled, characters will not be echoed until this command has been completely processed, i.e., the next programming message will be echoed

If you call **echo** without an argument, the GPIB-MAC returns the current setting.

Examples:

Note: The following examples show commands as you would enter them at a terminal.

1. echo 1<CR> Turn on character echoing.

2. ECHO O<CR> 'Disable character echoing.

3. echo<CR> What is the current echo status?

response: 0<CR><LF> (character echo is disabled)

eos - Change/Disable GPIB EOS Termination Mode

eos: Initialization function

Syntax: eos [[R][X][B] eoschar]<CR>
or
eos D<CR>

Purpose: You use eos at the beginning of your program if you wish to use an eos mode when you transfer data to and from the GPIB. eos tells the GPIB-MAC when to stop reading information from the GPIB. eos also enables the GPIB-MAC to tell other devices that it is finished writing information to the GPIB. eos defines a specific character, end-of-string (EOS), to be recognized as a string terminator.

Remarks: The arguments R, X, B, and D specify GPIB termination methods. They enable or disable the corresponding eos mode. If a particular letter is specified, the corresponding eos mode is enabled. If it is not specified, the corresponding eos mode is disabled.

eoschar is a numeric string which represents a single ASCII character. For example, 10 represents the ASCII linefeed character.

DATA TRANSFER TERMINATION METHODS

Description	Letter
REOS - terminate read when EOS is detected.	R
XEOS - set EOI with EOS on write functions.	X
BIN - compare all 8 bits of EOS byte rather than low 7 bits (all read and write functions).	B
DISABLE - disable all eos modes.	D

Methods R and B determine how GPIB read operations performed by the GPIB-MAC terminate. If Method R alone is chosen, reads terminate when the low 7 bits of the byte that is read match the low 7 bits of the EOS character. If Methods R and B are chosen, a full 8-bit comparison is used.

Methods X and B together determine when GPIB write operations performed by the GPIB-MAC send the END message. If Method X alone is chosen, the END message is sent automatically with the EOS byte when the low 7 bits of that byte match the low 7 bits of the EOS character. If Methods X and B are chosen, a full 8-bit comparison is used.

Note that defining an EOS byte for the GPIB-MAC does not cause the GPIB-MAC to insert that byte into the data string when performing GPIB writes. To send the EOS byte, you must include it in the data string that you send following the **wrt** programming message.

By default, no eos modes are enabled.

If you call **eos** with B alone as an argument, the GPIB-MAC records the EARG error.

If you call eos without an argument, the GPIB-MAC returns to you the current eos settings.

The assignment made by this function remains in effect until you call eos again, call **onl,** or you turn off the GPIB-MAC.

Refer to the GPIB Read and Write Termination explanation in Section Three.

Examples:

```
1. PRINT #1,"eos R,B,10"        'Terminate read when <LF> is
                                'detected; compare all 8 bits; do
                                'not send EOI with <LF>.
   PRINT #1,"rd 10 5"           'Read 10 bytes from device 5 into
                                'serial port buffer.
   RESP$=INPUT$(10,#1)          'Input 10 bytes from serial port
                                'buffer.
   LINE INPUT #1,COUNT$         'Input string that indicates number
   CNT%=VAL(COUNT$)             'of bytes actually read from GPIB
   PRINT COUNT$;"bytes were read from GPIB"  'Print number
                                'of bytes that were read from the
                                'GPIB.


2. PRINT #1,"EOS X,13"          'Tell GPIB-MAC on wrt, send
                                'EOI with <CR>; on rd, do not
                                'terminate when <CR> is
                                'detected; compare 7 bits.
   PRINT #1,"wrt #10 5"         'GPIB-MAC sends EOI with
                                '<CR> (CHR$(13)) to tell
                                'listeners that this is the last byte
                                'of data.
   PRINT #1,"012345678"+CHR$(13)


3. PRINT #1,"eos"               'What are the current EOS
                                'settings?
   response: X,13<CR><LF>
```

eot - Enable/Disable END Message on GPIB Writes

eot: Initialization function

Syntax: **eot** [bool]<CR>

Purpose: You use **eot** at the beginning of your program if you wish to change how the GPIB-MAC terminates GPIB writes. Using **eot, you** tell the GPIB-MAC to automatically send or not send the GPIB END message with the last byte that it writes to the GPIB.

Remarks: If the argument **bool** is 1, the GPIB-MAC automatically sends the END message with the last byte of each **wrt. If the** argument bool is 0, END is not sent. The power-on default is 1.

If you call **eot** without an argument, the GPIB-MAC returns to you a 1 to indicate END termination is currently enabled, or a 0 to indicate END termination is currently disabled

The assignment made by **eot** remains in effect until you call **eot** again, call **onl,** or you turn off the GPIB-MAC.

The GPIB-MAC sends the END message by asserting the GPIB EOI signal during the last byte of a data transfer. **eot** is used primarily to send variable length data.

Refer to the GPIB Read and Write Termination explanation in Section Three.

Examples:

1. PRINT #1,"eot 0" ‘Disable END termination.

2. PRINT #1,"EOT 1” ‘Send END with last byte.
   PRINT #1,"WRT 3” Write data to device at address 3.
   PRINT # 1," ABCDE” The EOI line is automatically
   ‘asserted when the last byte (the
   letter E) is sent to tell the
   ‘Listeners it is the last byte of data.

3. PRINT #1,"eot" What is the current EOT setting?

response: 1<CR><LF> (END termination is currently enabled)

gts - Go from Active Controller to Standby

gts: Specialized Controller function

Syntax: gts [bool]<CR>

Purpose: You use **gts** to change the GPIB-MAC from Active Controller to Standby Controller. You use **gts** when the I/O and bus management functions do not meet the needs of your device. For example, you use **gts** if you wish to allow two external devices to talk to each other directly. The GPIB-MAC can selectively participate in the handshake of the data transfer and hold off the handshake when it detects the END message. The GPIB-MAC can then take control synchronously without possibly corrupting the transfer.

Remarks: If the argument **bool** is 1, shadow handshaking is enabled. If the argument **bool** is 0, shadow handshaking is not performed.

If you call gts without an argument, the GPIB-MAC returns to you the current controller status: CSB,O if the GPIB-MAC is in Standby without shadow handshaking; CSB,1 if the GPIB-MAC is in Standby with shadow handshaking; CAC if the GPIB-MAC is CIC but is not in Standby, i.e., it is the Active Controller; and CIDLE if the GPIB-MAC is not the CIC, i.e., is an IDLE Controller.

gts causes the GPIB-MAC to go to the Controller Standby state and to unassert the ATN signal if it is initially the Active Controller. gts permits GPIB devices to transfer data without the GPIB-MAC participating in the transfer.

If you enable shadow handshaking, the GPIB-MAC participates in the data handshake as an Acceptor without actually reading the data It monitors the transfers for the END (EOI or end-of-string character) message and holds off subsequent transfers. This mechanism allows the GPIB-MAC to take control synchronously on a subsequent operation such as **cmd** or **rpp.**

Before performing a **gts** with a shadow handshake, you should call eos to establish the proper end-of-string character or to disable the EOS detection if the end-of-string character used by the talker is not known.

If you call **gts** with an argument and the GPIB-MAC is not CIC, the GPIB-MAC records the ECIC error.

Refer also to **cac.**

Examples:

1. PRINT #1,"gts 0" 'GTS without shadow handshaking.

2. PRINT #1,"GTS 1" 'GTS with shadow handshaking.

3. PRINT #1,"gts" What is the standby status?

response: CSB,1<CR><LF> (GPIB-MAC is in standby status with shadow handshaking)

idMAC - Identify System

idMAC: General Use function

Syntax: **idMAC<CR>**

Purpose: You use **idMAC if you** wish to know the revision level of your software, or if you wish to know how much RAM is installed in your GPIB-MAC.

Remarks: The identification is returned in three strings. The first two strings identify the company product model, the software revision level, and a copyright notice. The third string identifies the number of bytes of RAM in the GPIB-MAC.

Example:

```
PRINT # 1 ,"idMAC"          'Get system identification
```

response: GPIB-MAC, Rev. A.0<CR><LF>
(c)1985 National Instruments<CR><LF>
2K bytes RAM<CR><LF>

ist - Set or Clear Individual Status Bit

ist: Parallel Poll function

Syntax: ist [bool]<CR>

Purpose: You use ist when the GPIB-MAC participates in a parallel poll that is conducted by another device that is Active Controller.

Remarks: If the argument **bool** is 1, the GPIB-MAC's individual status bit is set to 1. If the argument **bool** is 0, the GPIB-MAC's individual status bit is cleared. The power-on default is 0.

If you call ist without an argument, the GPIB-MAC returns the value of its individual status bit.

Refer also to **ppc** and Appendix F.

Examples:

1. PRINT #1,"ist 1" 'Set ist to 1.

2. PRINT #1,"IST 0" 'Clear ist to 0.

3. PRINT #1,"ist" 'What is ist set to?

response: 0<CR><LF> (ist is currently 0)

loc - Go to Local *

loc: Bus Management function

Syntax: loc [alist] <CR>

Purpose: You use **loc** to put a device in local program mode. In this mode you can program the device from its front panel. Since a device must usually be placed in remote program mode before it can be programmed from the GPIB, the GPIB-MAC automatically puts the device in remote program mode. You then use **loc** to return devices to local program mode.

Remarks: The argument **alist** is a list of **addrs** separated by commas or spaces. **addrs are** device addresses that specify the GPIB addresses of the devices you wish to return to local mode.

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

If you call **loc** with **alist, the** GPIB-MAC places the specified device(s) in local mode using the Go To Local (GTL) command

If this is the first function you call that requires GPIB controller capability, and you have not disabled System Controller capability with **rsc,** the GPIB-MAC sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable. If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If you call **loc** without **alist,** and the GPIB-MAC is System Controller, the GPIB-MAC returns all devices to local mode by unasserting REN and asserting it again. If you call **loc** without **alist** and the GPIB-MAC is not System Controller, the GPIB-MAC records the ESAC error.

Refer to Appendix B for more error information.

Examples:

1. PRINT #1,"loc 6+22,4+23,7" 'Put 3 devices in local mode.

2. PRINT #1,"LOC" 'Put all devices in local mode.

* frequently used function

onl - Place the GPIB-MAC Online/Offline

onl: Initialization function

Syntax: **onl** [bool] <CR>

Purpose: You use **onl** to disable communications between the GPIB-MAC and the GPIB, or to reinitialize the GPIB-MAC characteristics to their default values.

Remarks: If the argument **bool** is 1, the GPIB-MAC places itself online. If the argument **bool** is 0, the GPIB-MAC places itself offline. By default, the GPIB-MAC powers up online, is in the Idle Controller state, and configures itself to be the System Controller.

If you call **onl** without an argument, the GPIB-MAC returns the current status of the GPIB-MAC, which is 0 if the GPIB-MAC is offline and 1 if the GPIB-MAC is online.

Placing the GPIB-MAC offline may be thought of as disconnecting its GPIB cable from the other GPIB devices.

Placing the GPIB-MAC online allows the GPIB-MAC to communicate over the GPIB, and also restores all GPIB-MAC settings to their power-on values.

Refer to the Serial Port Characteristics table and the GPIB Characteristics table in Section Three for the GPIB-MAC power-on settings.

Examples:

```
1. PRINT #1,"onl 1"          'Put the GPIB-MAC online and
                             'restore its power-on settings.

2. PRINT #1,"ONL 0"          'Put the GPIB-MAC offline to
                             'prevent it from communicating
                             'with the GPIB.
```

pct - Pass Control

pct: Specialized Controller function

Syntax: **pct addr<CR>**

Purpose: You use **pct** to pass Controller-In-Charge (CIC) authority from the GPIB-MAC to some other device.

Remarks: The argument **addr** is the address of the device you wish you wish to pass control to. **addr** consists of a primary address and an optional address. The secondary address is separated from the primary address by a plus sign (+). Both address are expressed as numeric strings.

Only the lower five bits of each address are significant. These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

pct passes CIC authority from the GPIB-MAC to the device specified by **addr. The** GPIB-MAC automatically goes to Controller Idle State. It is assumed that the target device has Controller capability.

If you call **pct** with an argument and the GPIB- MAC is not CIC, it records the ECIC error.

If you call **pct** without an argument, the GPIB-MAC records the EARG error.

Example:

```
PRINT #1,"pct 7+18"          'Pass control to device with
                             'primary address 7 and
                             'secondary address 18.
```

ppc - Parallel Poll Configure

ppc: Parallel Poll function

Syntax: **ppc** addr,ppr,s [addr,ppr,s] [addr,ppr,s]...<CR>

Purpose: You use **ppc** to configure specified devices to respond to parallel polls in a certain manner.

Remarks: **addr** specifies the GPIB address of the device to be enabled or disabled for parallel polls. **addr** consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+). Both addresses are expressed as numeric strings.

Only the lower five bits of each address are significant These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

The argument **ppr** is an integer string between 1 and 8 specifying the data line on which to respond.

The argument s is either 0 or 1 and is interpreted along with the value of the device's individual status bit to determine whether to drive the line true or false.

Each group of **addr,ppr,s** may be separated by either a comma or space, just as any list of arguments.

If you call **ppc** without an argument, the GPIB-MAC records the EARG error.

If this is the first function you call that requires GPIB controller capability, and you have not disabled System Controller capability with rsc, the GPIB-MAC sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

The GPIB-MAC takes the arguments **ppr** and s and constructs the appropriate parallel poll enable (PPE) message for each **addr** specified.

When **addr** is the address of the GPIB-MAC, the GPIB-MAC programs itself to respond to a parallel poll by setting its local poll enable (lpe) message to the value specified.

Refer also to **ist, ppu, rpp,** and to Appendix F on parallel polling.

Example:

```
PRINT #1,"PPC18+23,8,023+10,7,1" 'Configure 2
                                 'devices for parallel poll.
PRINT #1,"RPP"                   'Conduct a Parallel poll of 2
                                 'devices configured above.
response:192<CR><LF>             (both devices responded
                                 positively)

LINE INPUT #1,RESP$
PPR%=VAL(RESP$)                  'Assign parallel poll response to
                                 'integer variable.
```

ppu - Parallel Poll Unconfigure

ppu: Parallel Poll function

Syntax: **ppu** [alist]<CR>

Purpose: You use **ppu** if you are performing parallel polls and you wish to prevent certain devices from responding.

Remarks: The argument **alist** is a list of addrs which are separated by commas or spaces. addrs are device addresses that specify the GPIB addresses of the device or devices to be disabled from parallel polls.

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

If you call **ppu** with **alist,** the GPIB-MAC unconfigures from parallel polls only those devices specified in **alist.**

If you call **ppu** without **alist,** the GPIB-MAC unconfigures all devices from parallel polls.

If this is the first function you call that requires GPIB controller capability, and you have not disabled System Controller capability with **rsc**, the GPIB-MAC sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If the address of the GPIB-MAC is included in **alist**, the GPIB-MAC disables itself from responding to parallel polls.

Refer also to **ist**, **ppc**, **rpp**, and to Appendix F on parallel polls.

Examples:

1. PRINT #1,"ppu 14" 'Send the PPU command to device 14.

2. PRINT #1,"PPU" 'Send the PPU command to all devices.

rd - Read Data *

rd: I/O function

Syntax: **rd** #count [addr]<CR>

Purpose: You use **rd** to read data from the GPIB.

Remarks: The argument **#count** is a numeric string preceded by a number sign (#). **#count** specifies the number of bytes to read. **count** must not contain a comma. It can specify a number between 1 and 65535.

The argument **addr** is a device address that specifies the address of the device to be addressed as a Talker. **addr** consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

The GPIB-MAC reads data from the GPIB until the specified byte count is reached, the GPIB END message is received with a data byte, the EOS byte is received, or a timeout occurs.

Refer also to eos, **eot,** and **tmo.**

Because you may not know for certain the number of bytes actually read from the GPIB, the GPIB-MAC returns the received GPIB data to you as follows. First, the GPIB-MAC returns to you all bytes it read from the GPIB. Next, it sends null bytes until the total number of bytes returned to you matches your requested count. Finally, it returns a numeric string representing the number of bytes that it actually read from the GPIB.

For example, if you send the GPIB-MAC the programming message "rd 10"<CR>, it reads data from the GPIB until it receives 10 bytes of data, the END message, or an eos byte. Let's say the GPIB-MAC receives END with the fourth data byte. The GPIB-MAC then returns to you the four data bytes, followed by 6 null bytes, followed by an ASCII 4 and <CR><LF>. A null byte is decimal 0. You should always read back **count** bytes of data from the serial port, then look at the remaining bytes to determine how many of the **count** bytes were read from the GPIB. Refer to the example at the end of this description.

The GPIB-MAC aborts the GPIB read and records the EABO error if, at any time during the GPIB read, the time limit set for I/O functions expires. This limit is 10 seconds unless you use **tmo** to change it.

If the GPIB-MAC is CIC, **rd** will cause the GPIB-MAC to address itself to Listen if it is not already addressed. If you specify the address of the Talker, the GPIB-MAC will also address that device to Talk. If you do not specify the address of the Talker, the GPIB-MAC will assume that the Talker has already been addressed.

The GPIB-MAC then places itself in Standby Controller state with ATN off and remains there after the read operation is completed

If you specify an address, the GPIB-MAC must be CIC to perform the addressing.

If this is the first function you call that requires GPIB controller capability, and you have not disabled System Controller capability with rsc, the GPIB-MAC sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If the GPIB-MAC is not CIC and you do not specify the talker address, the GPIB-MAC assumes it will be addressed by the Controller, then proceeds.

If you call **rd** without an argument, the GPIB-MAC records the EARG error.

Refer also to **tmo.**

Example:

```
PRINT #1,"rd #10 3"          Read up to 10 bytes from the
                             'GPIB device at address 3.
RESP$=INPUT$(10,#1)          'Input 10 bytes from serial port
                             'buffer.
LINE INPUT #1,COUNT$         'Input ASCII string representing
                             'number of bytes read from the
                             'GPIB.
COUNT%=VAL(COUNT$)           'COUNT% is number of bytes
                             'read from GPIB; remaining bytes
                             'in RESP$ may be ignored.
```

* frequently used function

rpp - Request (Conduct) a Parallel Poll

rpp: Parallel Poll function

Syntax: **rpp<CR>**

Purpose: **You use rpp if** you wish to conduct a parallel poll to obtain information from several devices at the same time.

Remarks: rpp causes the GPIB-MAC to conduct a parallel poll of previously configured devices by sending the IDY message (ATN and EOI both asserted) and reading the response from the GPIB data lines. The GPIB-MAC pulses the IDY message for greater than or equal to 2 microseconds and expects valid responses within that time. It remains Active Controller afterward

The GPIB-MAC returns the Parallel Poll Response (PPR) following the poll in the form of a numeric string representing the decimal value of the response.

If this is the first function you call that requires GPIB controller capability, and you have not disabled System Controller capability with **rsc,** the GPIB-MAC sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

Refer also to ist, **ppc, ppu,** and to Appendix F on parallel polls.

Example:

```
PRINT #1,"ppc13,1,015,3,0"+CHR$(13)+"rpp" 'Configure 2
                                          'devices for parallel polls and poll
                                          'them.
```

response: 5<CR><LF> (both devices responded positively)

```
LINE INPUT #1,RESP$
PPR%=VAL(RESP$)          'Get parallel poll response from
                         'serial port buffer and assign it to
                         'integer variable PPR%.
PRINT # 1,' 'ppu"        'Unconfigure all devices from
                         'parallel polls.
```

rsc - Request or Release System Control

rsc: Initialization function

Syntax: **rsc** [bool] <CR>

Purpose: You use rsc if some other device in your GPIB system should be System Controller.

Remarks: If the argument **bool** is 1, the GPIB-MAC configures itself to be the GPIB System Controller. If the argument **bool** is 0, the GPIB-MAC configures itself as not System Controller.

If you call **rsc** without an argument, the GPIB-MAC returns to you its System Controller status, which is 0 if GPIB-MAC is not currently System Controller or 1 if the GPIB-MAC is System Controller.

As System Controller the GPIB-MAC can send the Interface Clear (IFC) and Remote Enable (REN) messages to GPIB devices. If some other Controller asserts Interface Clear, the GPIB-MAC cannot respond unless it is configured as not System Controller.

In most applications, the GPIB-MAC will be System Controller. In some applications, the GPIB-MAC will never be System Controller. In either case, **rsc** is used only if the Macintosh is not going to be System Controller while the program executes. The IEEE-488 standard does not specifically allow schemes in which System Control can be passed from one device to another, however, **rsc** could be used in such a scheme.

The GPIB-MAC configures itself to be System Controller at power-on.

Refer also to **sic** and **sre.**

Examples:

1. PRINT #1,"rsc 1" ‘Enable GPIB-MAC to be system ‘controller.

2. PRINT #1,"rsc 0" ‘Disable system control.

3. PRINT #1,"rsc" What is the current system ‘controller status?

response: 0<CR><LF> (GPIB-MAC is not the System Controller)

rsp - Request (Conduct) a Serial Poll

rsp: Serial Poll function

Syntax: **rsp** alist<CR>

Purpose: You use **rsp if you** wish to conduct a serial poll to obtain device-specific status information from one or more devices.

Remarks: **The** argument **alist** is a list of addrs which are separated by commas or spaces. **addrs are** device addresses that specify the GPIB addresses you wish to poll.

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 **or** 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

rsp serially polls the specified devices to obtain their status bytes. If bit 6 (the hex 40 or RQS bit) of a device's response is set, its status response is positive, i.e., that device is requesting service. Before **rsp** completes, all devices are unaddressed.

The interpretation of each device's response, other than the RQS bit, is device specific. For example, the polled device might set a particular bit in the response byte to indicate that it has data to transfer, and another bit to indicate a need for reprogramming. Consult the device documentation for interpretation of the response byte.

Each device's serial poll response byte is returned as a numeric string giving the decimal value of the byte, followed by <CR> and <LF>. If a device does not respond in the timeout period, the GPIB-MAC returns string -1 and records the EABO error. The time limit is set to 1/10 second unless you called **tmo** to change it. Each response corresponds directly to an address you specify, therefore, there are exactly as many lines of responses, including - 1, as the number of addresses you specify.

If you call **rsp** and the GPIB-MAC is not CIC, it attempts to become CIC. If it cannot become CIC, it records the ECIC error. Refer to Appendix B for more information.

If this is the first function you call that requires GPIB controller capability, and you have not disabled System Controller capability with rsc, the GPIB-MAC sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If you call **rsp** without an argument, the GPIB-MAC records the EARG error.

Refer also to **tmo** for timeout information.

Example:

```
PRINT #1,"rsp 1+28,5,9" 'Poll 3 devices.

response: 42<CR><LF>  (device 9 did not respondwithin
          30<CR><LF> the timeout period)
          -1<CR><LF>

DIM SPR%(2)                    Read 3 responses from serial port
                               'buffer.
FOR I=0 to 2
LINE INPUT #1,RESP$            'Store each serial poll response in
SPR%(I)=VAL(RESP$)             'the array SPR%.
IF SPR%(I) = -1 THEN GOSUB 1000 '1000 is an error
                               'routine.
NEXT1
REM Code will now interpret serial poll responses.
```

rsv - Request Service/Set or Change Serial Poll Status Byte

rsv: Serial Poll function

Syntax: **rsv** [spbyte] <CR>

Purpose: You use rsv if the GPIB-MAC is not GPIB Controller and you wish to request service from the Controller using the Service Request (SRQ) signal. The GPIB-MAC will provide a user defined status byte when the Controller serially polls it.

Remarks: The argument spbyte is a numeric string specifying the decimal value of the new GPIB-MAC serial poll response byte.

The serial poll response byte is the status byte the GPIB-MAC provides when serially polled by another device that is CIC. If bit 6 (hex 40 RQS bit) is also set, the GPIB-MAC additionally requests service by asserting the SRQ line.

If you call rsv without an argument, the GPIB-MAC returns a numeric string containing the decimal value of its serial poll status byte.

Examples:

1. PRINT # 1 ,"rsv \x46"	Request service with serial poll ‘response = 6.
2, PRINT #1,"rsv"	What is the current serial poll ‘status byte?
response: 70<CR><LF>	(The current status byte=decimal 70 or hex 46)

sic - Send Interface Clear

sic: Specialized Controller function

Syntax: sic [time] <CR>

Purpose: You use sic if the initialization, I/O, and bus management functions do not meet the needs of your device, and you wish more precise control over the GPIB. **sic** makes the GPIB-MAC CIC and initializes the GPIB. sic is not a function you will use frequently because in most cases the first I/O or bus management function you call will do this automatically.

Remarks: The argument **time** is a numeric string specifying any number of seconds between .0001 and 3600, which corresponds to time limits between 100 microseconds and 1 hour. time must not contain a comma.

If you call sic without an argument, IFC is sent for 500 microseconds. The action of asserting IFC for at least 100 microseconds initializes the GPIB and makes the interface board become CIC. When needed, **sic** is generally used at the beginning of a program to make the GPIB-MAC CIC and is used when a bus fault condition is suspected.

The IFC signal resets only the GPIB interface functions of bus devices and not the internal device functions. Device functions are reset with the **clr** programming message. To determine the effect of these messages, consult the device documentation.

If you are in a debugging environment, you may want to vary the amount of time IFC is asserted. For example, you may set time to 10 seconds to allow you to check on a bus analyzer that IFC is actually being asserted. Otherwise, you do not need to include the **time** argument.

The GPIB-MAC records the ESAC error if you have disabled its System Controller capability with the **rsc** function. It records the EARG error if you specify a time outside the range .0001 to 3600.

Refer also to clr and to Appendix D.

Examples:

```
1. PRINT #1,"sic"          'Send interface clear for 500
                           'microseconds.

2. PRINT #1,"SIC.01"       'Send interface clear for 10
                           'milliseconds.
```

spign - Ignore Serial Port Errors

spign: Serial Port function

Syntax: spign [bool] <CR>

Purpose: You use spign at the beginning of your program if you wish to change the effect that serial port errors have on how the GPIB-MAC processes programming messages and data. This function tells the GPIB-MAC to ignore or not to ignore the occurrence of serial port errors. By default, the GPIB-MAC ignores serial port errors.

Remarks: If the argument **bool** is 0, the GPIB-MAC will not ignore serial port errors. When **bool** is 0, the GPIB-MAC does not execute programming messages that contain serial port errors. A list of serial port errors are listed in Appendix B. Also, if a serial port error occurs with any byte contained in a **cmd** or **wrt** data string, the GPIB-MAC discards that data byte and all remaining bytes in the string.

The serial port errors include parity, overrun, framing, and overflow errors.

If the argument **bool** is 1, the GPIB-MAC executes all programming messages and sends all data, even if serial port errors occur as the messages and data bytes are received

No matter what value **bool** has, the GPIB-MAC still records the errors in the serial-error portion of the status area.

If you call **spign** without an argument, the GPIB-MAC returns to you the current setting.

Refer also to **cmd** and **wrt.**

Examples:

```
1. PRINT #1,"spign 0"      'Do not execute programming
                           'messages or process data that
                           'contain serial port errors.

2. PRINT #1,"spign 1"      'Execute all programming messages
                           'and send all data, even if serial port
                           'errors occur.
```

sre - Set or Clear Remote Enable

sre: Specialized Controller function

Syntax: sre [bool]<CR>

Purpose: You use sre if the I/O and bus management functions do not meet the needs of your device. **sre** gives you more precise control over the GPIB. Use **sre** to turn the Remote Enable signal on and off. **sre** is not a function you will use frequently because in most cases, the first I/O or bus management function you call will set remote enable automatically.

Remarks: If the argument **bool** is 1, the GPIB-MAC asserts the Remote Enable (REN) signal. If the argument **bool** is 0, the GPIB-MAC unasserts REN.

Many GPIB devices have a remote program mode and a local program mode. It is usually necessary to place devices in remote mode before programming them from the GPIB. A device enters the remote mode when the REN line is asserted and the device receives its listen address.

Use **cmd** to send a device its listen address after using **sre.** Use **loc** to return the device to local program mode.

If you call **sre with** an argument and the GPIB-MAC is not System Controller, the GPIB-MAC records the ESAC error.

If you call **sre** without an argument, the GPIB-MAC returns its current remote status: 1=remote, 0=local.

Refer also to **rsc, cmd,** and **loc**.

Examples:

1. PRINT #1,"SRE 1" 'Set REN.

2. PRINT #1,"sre 0" 'Unassert REN.

stat - Return GPIB-MAC Status

stat: General Use function

Syntax: stat [[c] n]<CR>
or
stat [c] s<CR>
or
stat [c] n s<CR>

Purpose: You use **stat** to obtain the status of the GPIB-MAC to see if certain conditions are currently present. You use stat most often to see if the previous operation resulted in an error.

Remarks: You should use stat frequently in the early stages of your of your program development when your device's responses are likely to be unpredictable. The GPIB-MAC responds with status information in a form depending on the mode or combination of modes you chose. **n** indicates that the status information will be returned as numeric strings. s indicates that the status information will be returned in symbolic format, i.e., as mnemonic strings. c specifies that the status will be returned after each programming message, eliminating the need to call **stat** after each programming message.

Normally, you use s only when you are debugging your code and you want to print the mnemonic for each piece of status information.

The status information returned by the GPIB-MAC contains four pieces of information: the GPIB-MAC status, a gpib-error code, a serial-error code, and a count. The GPIB-MAC returns a <CR><LF> following each piece of the response.

Status represents a combination of GPIB-MAC conditions. Internally in the GPIB-MAC, status is stored as a 16-bit integer. Each bit in the integer represents a single condition. A bit value of 1 indicates that the corresponding condition is in effect; a bit value of zero indicates that the condition is not in effect. Since more than one GPIB-MAC condition may exist at one time, more than one bit may be set in status. The highest order bit of status, also called the sign bit, is set when the GPIB-MAC detects either a GPIB error or a serial port error. Consequently, when status is negative, an error condition exists, and when status is positive, no error condition exists.

gpib-error represents a single GPIB error condition present.

serial-error represents a single serial error condition present.

count is the number of bytes transferred over the GPIB by the last **rd, wrt,** or **cmd** function.

GPIB STATUS CONDITIONS

Bit	Numeric Value (n)	Symbolic Value (s)	Description
15	- 32768	ERR	Error detected
14	16384	TIMO	Timeout
13	8192	END	EOI or EOS detected
12	4096	SRQI	SRQ detected while CIC
11	2048		Reserved
10	1024		Reserved
9	512	-	Reserved
8	256	CMPL	Operation completed
7	128	LOK	Lockout state

GPIB STATUS CONDITIONS (CONTINUED)

Bit	Numeric Value (n)	Symbolic Value (s)	Description
6	64	REM	Remote state
5	32	CIC	Controller-In-Charge
4	16	ATN	Attention asserted
3	8	TACS	Talker active
2	4	LACS	Listener active
1	2	DTAS	Device trigger state
0	1	DCAS	Device clear state

GPIB ERROR CONDITIONS

Numeric Value (n)	Symbolic Value (s)	Description
0	NGER	No GPIB error condition to report
1	ECIC	Command requires GPIB-MAC to be CIC
2	ENOL	Write detected no listeners
3	EADR	GPIB-MAC not addressed correctly
4	EARG	Invalid argument or arguments
5	ESAC	Command requires GPIB-MAC to be SC
	EABO	I/O operation aborted
F-16	-	Reserved
17	ECMD	Unrecognized command

SERIAL PORT ERROR CONDITIONS

Numeric Value (n)	Symbolic Value (s)	Description
0	NSER	No serial port error condition to report
1	EPAR	Serial port parity error
2	EORN	Serial port overrun error
3	EOFL	Serial port receive buffer overflow
4	EFRM	Serial port framing error

A detailed description of the conditions under which each bit in status is set or cleared may be found in Appendix **B**.

In general, the GPIB-MAC updates the first three status variables at the end of each programming message. It updates the fourth, count, after a cmd, rd, or **wrt** function The errors reported correspond to the previous programming message. For example, if you call wrt and then stat s, any errors returned to you correspond to errors in the **wrt** programming message, not **stat.** However, if status is returned in continuous mode, the status information corresponds to the current programming message. For example, suppose you call stat c **s** to set up continuous status reporting. After reading the status information returned for the **stat call, you call wrt. The** GPIB-MAC then returns the status information that corresponds to the wrt message.

Refer to the following examples for ways in which to make use of the status information.

When you wish to begin continuous status reporting, send the stat c s, stat c n, or stat c **n s** programming message. Status information will be immediately returned indicating the current status conditions. When

you call **stat** with both s and **n the** numeric status is always returned first.

If you call stat without an argument, continuous status reporting is disabled.

Notice that when you send several programming messages to the GPIB-MAC, it buffers them and processes each one without any delay in between. However, if you enable continuous status reporting and check the status of each programming message before sending the next, the GPIB-MAC waits for each subsequent programming message to arrive at the serial port before processing it. This slows down the overall performance of your program. If speed is a primary concern, disable continuous status reporting.

Examples:

1.
```
10   PRINT #1,"stat n"      'Get GPIB-MAC status.
20   REM GPIB-MAC responds with:
30 REM 340<CR><LF>0<CR><LF>0<CR><LF>0<CR><LF>
40   REM Now read status into variables.
50   STATUS% = VAL(LINE INPUT#1,STATUS$)
60   LINE INPUT # 1 ,GPIBERR$
70   LINE INPUT #1,SPERR$
80   LINE INPUT #1,COUNT$
90   REM Go to error routine at 500 if error occurred.
100  IF STATUS% < 0 THEN GOT0 500
110  REM Go to SRQ service routine if SRQ is asserted.
120  IF (STATUS% AND &H1000) THEN GOT0 400
...
400  REM
410  REM Place code here to service SRQ.
420  REM
500  REM Print gpib-error and serial-error values to
5 10 REM determine what errors occurred
520 PRINT "GPIB-error = ";GPIBERR$
530 PRINT "Serial-error = ";SPERR$
540  STOP
```

2. 10 PRINT #1, "stat s"
   20 REM If it has just read 3 bytes from the GPIB,
   30 REM GPIB-MAC responds with:
   40 REM CMPL,REM,ATN,LACS<CR><LF>NGER<CR><LF>
   50 REM NSER<CR><LF>3<CR><LF>

3. The following list illustrates what appears on the screen when you are programming the GPIB-MAC from a terminal. Programming messages you enter are in normal type. GPIB-MAC responses are in bold. The statements in parentheses are comments.

```
stat c s n                    (enable continuous status reporting)
344

8
3
CMPL,REM,ATN,TACS             (status returned)
NGER
NSER
3
wrt 10
ABCDE                         (Write the string ABCDE)
344                           (device 10.)
0                             (Status returned.)
0
5
CMPL,REM,ATN,TACS
NGER
NSER
5
```

tmo - Change or Disable Time Limit

tmo: Initialization function

Syntax: tmo [timeio] [,timesp] <CR>

Purpose: You use tmo at the beginning of your program to change the time limits in effect on the GPIB-MAC. The time limits prevent the GPIB-MAC from hanging indefinitely when an error situation prevents normal completion of an operation.

Remarks: The arguments **timeio** and **timesp are** numeric strings. **timeio** specifies the amount of time in seconds the GPIB-MAC waits for an I/O operation **(rd, wrt, cmd)** or the **wait** function to complete. **timesp** specifies the amount of time in seconds each device is given in which to respond to a serial poll. The power-on timeouts are 10 seconds for **timeio** and 1/10 of a second for timesp.

timeio and timesp may be any decimal number between .00001 and 3600 which corresponds to time limits between 10 microseconds and 1 hour. 10,.1 specifies a time of 10 seconds for I/O operations and 1/10 of a second for serial poll response. timeio and timesp may also be 0, which disables either timeout accordingly. Neither timeio nor timesp may contain commas (e.g., 1000 is correct but 1,000 is not).

The timeio time limit is in effect for the cmd, rd, and **wrt** functions. If the GPIB-MAC cannot complete any of these functions within the period of time set by timeio, it aborts the function and records the EABO error. Bytes that were transferred before the timeout are not affected.

The timeio time limit is also the maximum amount of time the wait function waits when you call it with the TIMO bit set in the wait mask.

The timesp time limit is in effect only for the **rsp** function. If a polled device fails to respond within the amount of time indicated by **timesp, the** GPIB-MAC returns an error flag.

Refer to the **rsp** programming message.

If you want to change only the timeout value for serial polls, a comma must precede the serial poll timeout value.

If you call **tmo** without an argument, the GPIB-MAC returns a numeric string representing the current timeout settings. It records the EARG error if you specify a time value outside the range .00001 to 3600.

The assignment made by this function remains in effect until you call **tmo** again, call **onl,** or turn off the GPIB-MAC.

Examples:

1. PRINT #1,"tmo 30" 'Set timeout for I/O operations to 30 'seconds; leave serial poll timeout 'unchanged.

2. PRINT #1,"TMO" 'Print current timeout settings.

response: 30,.1<CR><LF>

3. PRINT #1,"tmo ,1" 'Set serial poll timeout for one second; 'leave I/O timeout unchanged.

trg - Trigger Selected Device(s) *

trg: Bus Management function

Syntax: **trg** alist<CR>

Purpose: You use **trg** to trigger the specified devices. The instructions for each GPIB device explain when you should trigger them and what effect the trigger has.

Remarks: The argument **alist** is a list of **addrs** separated by commas or spaces. **addrs are** device addresses that specify the GPIB addresses you wish to trigger.

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

If you call **trg** without an argument, the EARG error is posted.

If this is the first function you call that requires GPIB controller capability, and you have not disabled System Controller capabiltity with **rsc, the** GPIB-MAC sends Interface Clear (IFC) to make itself CIC. It also asserts Remote Enable.

If you passed control to some other GPIB **device,control** must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

Example:

```
PRINT #1,"trg 2+10,4,5+7"        'Trigger 3 devices.
```

* frequently used function

wait - Wait for Selected Event

wait: General Use function

Syntax: wait mask<CR>

Purpose: You use **wait** to monitor selected GPIB events and to delay any further GPIB-MAC activity until one of them occurs.

Remarks: The argument **mask** is a numeric string which specifies the events to wait for. The numeric string represents a bit mask containing a subset of the same bit assignments as the status word described in the **stat** function. Each bit is set or cleared to wait or not to wait, respectively, for the corresponding event to occur. The numeric string may be expressed as decimal, octal, or hexadecimal.

After receiving the **wait** programming message, the GPIB-MAC monitors GPIB activity. When any event corresponding to the bits set in **mask occurs, the GPIB-MAC** returns status information indicating its current status. If continuous status reporting has been enabled, status will be reported in the format requested. If continuous status has not been enabled, status will be returned in numeric format.

You could use **wait,** for example, if you wish to wait until a device requests service before you perform a serial poll. In this case you send the **wait** programming message with **mask=4096,** then wait for status information to be returned. You then check that status to see if the SRQI bit is set in the returned status indicators.

To prevent the GPIB-MAC from waiting indefinitely for SRQ to be asserted, set the SRQI and **TIMO** bits by setting the **mask** equal to 4096 + 16384. This will cause the wait to terminate either on SRQI or TIMO, whichever occurs first.

It is recommended that you always include the TIMO bit value in mask when you wait for an event.

WAIT MASK VALUES

Bit	Hex Value	Decimal Value	Mnemonic	Description
15			-	Reserved
14	4000	16384	TIMO	Timeout
13		-	-	Reserved
12	1000	4096	SRQI	SRQ detected while CIC
11		-		Reserved
10	-	-	-	Reserved
9			-	Reserved
8	-			Reserved
7	80	128	LOK	Lockout state
6	40	64	REM	Remote state
5	20	32	CIC	Controller-In-Charge
4	10	16	ATN	Attention asserted
3	8	8	TACS	Talker active
2	4	4	LACS	Listener active
1	2	2	DTAS	Device trigger state
0	1	1	DCAS	Device clear state

If mask=0 the function completes immediately,

If the TIMO bit is 0 or the **timeio time limit** is set to 0 with **tmo,** timeouts for this function are disabled. You should disable timeouts only when you are certain the selected event will occur; otherwise, the GPIB-MAC waits indefinitely for the event to occur.

If you call **wait** without an argument, the GPIB-MAC records the EARG error.

Refer also to **stat** and **tmo.**

Examples:

```
1. PRINT #1,"wait \x5000" 'Wait for TIMO or SRQI.
   STATUS%=VAL(LINE INPUT #1,STATUS$)'Get status info.
   GPIBERR%=VAL(LINE IN-PUT # 1 ,GPIBERR$)
   SPERR%=VAL(LINE INPUT #1,SPERR$)
   COUNT%=VAL(LINE INPUT #l,COUNT$)
   IF (STATUS% AND &H4000) <> 0 THEN GOT0 1000
                        'If TIMO bit is set we timed out before
                        'getting SRQI. Go to an error routine
                        'at line 1000.
   IF(STATUS% AND &H1000) <> 0 THEN GOT0 200
                        'If SRQI bit set, go to routine to
                        'conduct a serial poll.


2. PRINT #1,"wait 4"     'Wait indefinitely to become LACS.
   STATUS%=VAL(LINE INPUT #1,STATUS$)'Get status info.
   GPIBERR%=VAL(LINE INPUT #1 ,GPIBERR$)
   SPERR%=VAL(LINE INPUT #1,SPERR$)
   COUNT%=VAL(LINE INPUT #1,COUNT$)
   PRINT #1,"rd10"       'Now that GPIB-MAC is addressed to
                         'listen, read 10 bytes from the GPIB.
   RESP$=INPUT$( 10,#1) 'Input 10 bytes from serial port
                         'buffer.
   LINE INPUT # 1 ,CNT!§ 'Input number of valid bytes in CNT$.
```

wrt - Write Data *

wrt: I/O function

Syntax: wrt [#count][alist]<CR>
data<CR>

Purpose: You use **wrt** to send data over the GPIB.

Remarks: The argument count is a numeric string preceded by a number sign (#). The string specifies a number between 1 and 65535 and must not contain a comma. **#count** specifies the number of data bytes to send. The number of data bytes must not include the carriage return that indicates the end of the programming message.

The argument **data** is a string of 8-bit characters which are transferred without any translation to the GPIB.

The argument **alist** is a list of **addrs** separated by commas or spaces. **addrs are** addresses that specify the GPIB addresses of the Listener (or Listeners, if more than one address is given).

A device address consists of a primary address and an optional secondary address. The secondary address is separated from the primary address by a plus sign (+).

Only the lower five bits of each address are significant. These bits may be in the range from 0 through 30 for both the primary and the secondary address. Therefore, the binary value 01100010 (decimal 98) is interpreted as decimal 2.

The following examples all specify a primary address of 0 and a secondary address of 2. The listen address is 32, the talk address is 64, and the secondary address is 2 or 98, which are equivalent.

0+2 or 0+98 or 32+98 or 0+\x62

When **#count** is not specified, the GPIB-MAC recognizes the end of the data string when it sees a carriage return or a line feed. **#count** is required when your data string contains embedded carriage return or linefeed characters.

If you specify an address list, the GPIB-MAC must be CIC to perform the addressing. If this is the **first** function you call that requires GPIB controller capability, and you have not disabled System Controller capability with **rsc,** the GPIB-MAC sends Interface Clear (IFC) to make itself CIC.

If you passed control to some other GPIB device, control must be passed back to you or you must send IFC to make yourself CIC before making this call. Otherwise, the ECIC error will be posted.

If you do not give an **alist** and the GPIB-MAC is not CIC, it assumes it will be addressed by the controller. If you do not give an **alist** and the GPIB-MAC is CIC, it addresses itself as talker and assumes the listeners are already addressed.

The first part of this programming message, up to <CR>, is buffered, meaning the GPIB-MAC will not act upon it until it receives <CR>. The string that follows the first line is piped to the GPIB, non-buffered. This allows you to send a string larger than the GPIB-MAC's internal buffer with one programming message.

The GPIB-MAC aborts the GPIB write if it receives Device Clear or Selected Device Clear and its Listen Address. The GPIB-MAC aborts the GPIB write and records the EABO error if, at any time during the GPIB write, the time limit set for I/O functions expires. This

limit is 10 seconds unless you use **tmo** to change it. The GPIB-MAC also aborts the **wrt** and records the ENOL error if there are no addressed Listeners when it begins to send data.

Refer also to **tmo** for timeout information, to Appendix B for more error information, and to **spign** for serial port error handling information

Examples:

1.
```
PRINT #1,"wrt #50 9+97"     Write 50 bytes to device at
FOR I = 1 TO50              'primary address 9 and secon-
PRINT #1,CHR$(A(I));        'dary address 97.
NEXT I
PRINT #1,CHR$(13);          'Send carriage return.
```

2.
```
PRINT #1,"wrt 2"            Write the data bytes ABCDE
PRINT #1,"ABCDE"            'at device at address 2.
```

* frequently used function

xon - Change Serial Port XON/XOFF Protocol

xon: Serial Port function

Syntax: xon [booltx][,boolrx]<CR>

Purpose: You use **xon** at the beginning of your program to configure the GPIB-MAC to communicate over the serial port using the same XON/XOFF protocol as your Macintosh.

Remarks: The argument **booltx** specifies whether to enable the XON/XOFF protocol when sending data out on the serial port. If the argument **booltx** is a 1, the GPIB-MAC monitors its serial receive buffer for XON/XOFF characters as it sends data over the serial port. If it receives the XOFF character (decimal 19 or <ctrl>s), it will immediately stop sending data. When it receives the XON character (decimal 17 or <ctrl>q), it begins sending data again.

If you want to send a data string that may contain a <ctrl>s or <ctrl>q, you must disable **booltx.**

The argument **boolrx** specifies whether to enable the XON/XOFF protocol when receiving data over the serial port. If the argument **boolrx** is a 1, and the GPIB-MAC is receiving data over the serial port, it sends XOFF over the serial port (if its serial receive buffer is almost full). This tells the sender to stop sending data. When the GPIB-MAC serial port receive buffer has room to safely receive more bytes, the GPIB-MAC sends XON over the serial port. This tells the sender to begin sending data again.

You should use XON/XOFF when you are transferring large amounts of data at a high baud rates. Without it, you are in danger of overflowing the GPIB-MAC internal buffer or your Macintosh's internal buffer.

The power-on default is that XON/XOFF for both cases is disabled.

If you call **xon** without an argument, the GPIB-MAC returns to you the current settings. (l=protocol enabled, O=protocol disabled)

Examples:

1. PRINT #1,"XON 1,1" 'Enable GPIB-MAC XON/XOFF 'protocol for TX and RX.

2. PRINT #1,"XON 0,1" 'Disable protocol on TX; 'enable protocol on RX

3. PRINT #1,"XON" Return current settings.

   response: 0,1<CR><LF> (transmit protocol disabled, receive protocol enabled)

4. PRINT #1, "XON ,0" 'Disable protocal on RX, keep 'current setting on TX.

Appendix A
Multiline Interface Messages

The following tables are multiline interface messages (Sent and Received with ATN TRUE).

Multiline Interface Messages

Hex	Octal	Decimal	ASCII	Message	Hex	Octal	Decimal	ASCII	Message
00	000	0	NUL		20	040	32	SP	MLA
01	001	1	SOH	GTL	21	041	33	!	MLA
02	002	2	STX		22	042	34	"	MLA
03	003	3	ETX		23	043	35	#	MLA
04	004	4	EOT	SDC	24	044	36	$	MLA
05	005	5	ENQ	PPC	25	045	37	%	MLA
06	006	6	ACK		26	046	38	&	MLA
07	007	7	BEL		27	047	39	'	MLA
08	010	8	BS	GET	28	050	40	(	MLA
09	011	9	HT	TCT	29	051	41	)	MLA
OA	012	10	LF		2A	052	42	*	MLA
OB	013	11	VT		2B	053	43	+	MLA
OC	014	12	FF		2c	054	44	,	MLA
OD	015	13	CR		2D	055	45	-	MLA
OE	016	14	SO		2E	056	46	.	MLA
OF	017	15	SI		2F	057	47	/	MLA
10	020	16	DLE		30	060	48	0	MLA
11	021	17	DC1	LLO	31	061	49	1	MLA
12	022	18	DC2		32	062	50	2	MLA
13	023	19	DC3		33	063	51	3	MLA
14	024	20	DC4	DCL	34	064	52	4	MLA
15	025	21	NAK	PPU	35	065	53	5	MLA
16	026	22	SYN		36	066	54	6	MLA
17	027	23	ETB		37	067	55	7	MLA
18	030	24	CAN	SPE	38	070	56	8	MLA
19	031	25	EM	SPD	39	071	57	9	MLA
1A	032	26	SUB		3A	072	58	:	MLA
1B	033	27	ESC		3B	073	59	;	MLA
1C	034	28	FS		3c	074	60	<	MLA
1D	035	29	GS		3D	075	61	=	MLA
1E	036	30	RS		3E	076	62	>	MLA
1F	037	31	US		3F	077	63	?	UNL

Multiline Interface Messages

Hex	Octal	Decimal	ASCII	Message	Hex	Octal	Decimal	ASCII	Message
40	100	64	@	MTA	60	140	96	‘	MS A,PPE
41	101	65	A	MTA	61	141	97	a	MS A,PPE
42	102	66	B	MTA	62	142	98	b	MS A,PPE
43	103	67	C	MTA	63	143	99	c	MS A,PPE
44	104	68	D	MTA	64	144	100	d	MS A,PPE
45	105	69	E	MTA	65	145	101	e	MS A,PPE
46	106	70	F	MTA	66	146	102	f	MS A,PPE
47	107	71	G	MTA	67	147	103	g	MS A,PPE
48	110	72	H	MTA	68	150	104	h	MS A,PPE
49	111	73	I	MTA	69	151	105	i	MS A,PPE
4A	112	74	J	MTA	6A	152	106	j	MS A,PPE
4B	113	75	K	MTA	6B	153	107	k	MS A,PPE
4c	114	76	L	MTA	6C	154	108	l	MS A,PPE
4D	115	77	M	MTA	6D	155	109	m	MS A,PPE
4E	116	78	N	MTA	6E	156	110	n	MS A,PPE
4F	117	79	O	MTA	6F	157	111	0	MS A,PPE
50	120	80	P	MTA	70	160	112	p	MS A,PPD
51	121	81	Q	MTA	71	161	113	q	MS A,PPD
52	122	82	R	MTA	72	162	114	r	MS A,PPD
53	123	83	S	MTA	73	163	115	s	MS A,PPD
54	124	84	T	MTA	74	164	116	t	MS A,PPD
55	125	85	U	MTA	75	165	117	u	MS A,PPD
56	126	86	V	MTA	76	166	118	v	MS A,PPD
57	127	87	W	MTA	77	167	119	w	MS A,PPD
58	130	88	X	MTA	78	170	120	x	MS A,PPD
59	131	89	Y	MTA	79	171	121	y	MS A,PPD
5A	132	90	Z	MTA	7A	172	122	z	MS A,PPD
5B	133	91	[	MTA	7B	173	123	{	MS A,PPD
5C	134	92	\	MTA	7c	174	124	|	MS A,PPD
5D	135	93	]	MTA	7D	175	125	}	MS A,PPD
5E	136	94	^	MTA	7E	176	126	~	MS A,PPD
5F	137	95	_	UNT	7F	177	127	DEL	

Appendix B - Status Information

This appendix describes the status and error information that the GPIB-MAC records as it executes each programming message. The number preceding each description is the numeric value of that bit in the status word or of the error code.

Status Bits

The following paragraphs describe the conditions represented by the bits in status.

ERR -32768

The ERR bit is set in **status** following any call that results in an error; the particular error may be determined by examining the **gpib-error** and **serial-error** values. The ERR bit is cleared following any call that does not result in an error.

Note: By examining this bit, you may check for an error condition after each call. An error made early in your application program may not become apparent until a later instruction. At that time, the error can be more difficult to locate.

TJMO 16384

The TIMO bit specifies whether a timeout has occurred. The TIMO bit is set in the status word following a call to **wait** if the TIMO bit of the **wait mask** parameter is also set and if the wait has exceeded the time limit value that is set by the **tmo call. The** TIMO bit is also set following a call to any of the I/O functions (e.g., **rd, wrt,** and **cmd),** if a timeout occurs during a call. The TIMO bit is cleared in the status word in all other circumstances.

END 8192

The END bit specifies whether the END or EOS message has been received. The END bit is set in the status word following a **rd** function if the END or EOS message was

detected during the read. While the GPIB-MAC is performing a shadow handshake as a result of the **gts** function, any other function call may return a status word with the END bit set if the END or EOS message occurred before or during that call. The END bit is cleared in the status word at the start of any subsequent programming message.

SRQI 4096
The SRQI bit specifies whether a device is requesting service. This bit is set in the status word whenever the SRQ line is asserted. The bit is cleared whenever the GPIB SRQ line is unasserted.

CMPL 256
The CMPL bit specifies that the operation relating to this status information is complete. This bit is always set, and is useful in identifying the status word from other responses.

LOK 128
The LOK bit specifies whether the GPIB-MAC is in a lockout state. The LOK bit is set whenever the GPIB-MAC detects the Local Lockout (LLO) message has been sent either by the GPIB-MAC or by another Controller. The LOK bit is cleared when the Remote Enable (REN) GPIB line becomes unasserted either by the GPIB-MAC or by another Controller.

REM 64
The REM bit specifies whether the GPIB-MAC is in remote state. The REM bit is set whenever the Remote Enable (REN) GPIB line is asserted and the GPIB-MAC detects its listen address has been sent either by the GPIB-MAC or by another Controller. The REM bit is cleared whenever REN becomes unasserted, or when the GPIB-MAC as a Listener detects the Go to Local (GTL) command has been sent either by the GPIB-MAC or by another Controller, or when the LOC function is called while the LOK bit is cleared in **status.**

CIC 32

The CIC bit specifies whether the GPIB-MAC is the Controller-In-Charge. The CIC bit is set whenever sic is called while the GPIB-MAC is System Controller, or when another Controller passes control to the GPIB-MAC. The CIC bit is cleared whenever the GPIB-MAC detects Interface Clear (IFC) from some other device that is System Controller, or when the GPIB-MAC passes control to another device.

ATN 16

The ATN bit specifies the state of the GPIB Attention (ATN) line. The ATN bit is set whenever the GPIB ATN line is asserted and cleared when the ATN line is unasserted.

TACS 8

The TACS bit specifies whether the GPIB-MAC has been addressed as a Talker. The TACS bit is set whenever the GPIB-MAC detects that its talk address (and secondary address, if enabled) has been sent either by the GPIB-MAC itself or by another Controller. The TACS bit is cleared whenever the GPIB-MAC detects the Untalk (UNT) command, a talk address other than its own, its own listen address, or Interface Clear (IX).

LACS 4

The LACS bit specifies whether the GPIB-MAC has been addressed as a Listener. The LACS bit is set whenever the GPIB-MAC detects that its listen address (and secondary address, if enabled) has been sent either by the GPIB-MAC itself or by another Controller. The LACS bit is also set whenever the GPIB-MAC shadow handshakes as a result of the **gts** function. The LACS bit is cleared whenever the GPIB-MAC detects that the Unlisten (UNL) command, its own talk address, Interface Clear (IFC), or gts is called without shadow handshake.

DTAS 2
The DTAS bit specifies whether the GPIB-MAC has detected a device trigger command. The DTAS bit is set whenever the GPIB-MAC as a Listener, detects the Group Execute Trigger (GET) command has been sent by another Controller. The DTAS bit is cleared in status at the start of any subsequent programming message.

DCAS 1
The DCAS bit specifies whether the GPIB-MAC has detected a device clear command The DCAS bit is set whenever the GPIB-MAC detects the Device Clear (DCL) command has been sent by itself or by another Controller, or whenever the GPIB-MAC as a Listener detects the Selected Device Clear (SDC) command has been sent by itself or by another Controller. The DCAS bit is cleared in **status** at the start of any subsequent programming message.

In addition to the above, the following situations also affect the bits in **status:**

* A call to the **onl** function clears the following bits:

  END LOK REM CIC TACS LACS DTAS DCAS

* A call to **onl** affects bits other than those listed here according to the rules explained for each bit.

GPIB Error Codes

When the ERR bit is set in **status,** a GPIB or a serial port error has occurred. The error code is indicated by **gpib-error or serial-error.**

The following paragraphs describe the **gpib-errors in** detail

NGER 0

The GPIB-MAC reports this error when GPIB-MAC detected no GPIB errors during the last operation.

ECIC 1

The GPIB-MAC records this error when you call a function that requires that the GPIB-MAC be CIC and it is not CIC.

In cases when the GPIB-MAC should always be the Controller-In-Charge, the remedy is to be sure to call **sic** to send Interface Clear before attempting any of these calls, and to avoid sending the command byte TCT (hex 09, Take Control). In multiple Controller-In-Charge situations, the remedy is to always be certain that the CIC bit appears in **status** before attempting these calls. If it is not, you may call **wait** (32) to wait to become CIC and to delay further processing until control is passed to the GPIB-MAC.

ENOL 2

The most common cause of this error is that the GPIB-MAC attempted to write to the GPIB when no listeners were addressed.

The remedy is to be sure that the proper listen address is in the **alist** argument string, to use **cmd** to properly address the listeners, or to be sure some other controller has addressed the listeners before you call wrt.

This error may occur more rarely in situations in which the GPIB-MAC is not the Controller-In-Charge and the Controller asserts ATN before the write call in progress has ended The remedy is either to reduce the write byte count to that which is expected by the Controller, or to resolve the situation on the Controller's end.

EADR 3

The GPIB-MAC records this error when it is not addressed to listen or to talk before read and write calls when it is not the Controller-In-Charge. The remedy is to be sure that the Controller addresses the GPIB-MAC to talk or listen before attempting the **wrt** or **rd.**

EARG 4

The GPIB-MAC records this error when you pass an invalid argument to a function call. The following are some examples:

tmo called with a value not in the range .OOOOl-3600
sic called with a value not in the range .OOOl-3600
eos called with meaningless termination method identifiers
caddr called with the value **31, 63, 95,** or 127.
ppc called with illegal parallel poll configurations

If your programming message contains more than one argument and you get this error, the GPIB-MAC discards all arguments and does not perform the function.

This also may be caused by a transmission error which corrupts the argument portion of the programming message or which corrupts the <CR> or <LF> that terminates the programming message. Use **stat** and **check serial-error** to determine if a transmission error has occurred

ESAC **5**

The GPIB-MAC records this error when **sic** or **sre is** called when the GPIB-MAC does not have System Controller capability. The remedy is to give the GPIB-MAC that capability by calling **rsc.** (At power on the GPIB-MAC assumes itself to be the System Controller.)

EABO 6

The GPIB-MAC records this error when I/O has been cancelled By far the most common cause of this error is a timeout condition.

To remedy a timeout error, if I/O is actually progressing but times out anyway, lengthen the timeout period with **tmo.** More frequently, however, the I/O is stuck (the Listener is not continuing to handshake or the Talker has stopped talking), or the byte count in the call which timed out was more than the other device was expecting. Be sure that both parties to the transfer understand what byte count is expected; or if possible, have the Talker use the END message to assist in early termination.

ECMD 17

The GPIB-MAC records this error when your programming message received by the GPIB-MAC does not contain a recognizable function name. This can happen if the function name is misspelled or if a transmission error occurred that resulted in the function name being corrupted. Check your function name spelling, and check **serial-error** to see if a serial port error has been posted.

Serial Port Error Codes

The following paragraphs describe the **serial port errors** in detail.

When a serial port error occurs as the GPIB-MAC receives a programming message, the GPIB-MAC posts the error and discards the message. If a serial port error occurs in the middle of a data stream following a **cmd** or **wrt** function, the GPIB-MAC discards that data byte and all subsequent data bytes. You may use the **spign** function to tell the GPIB-MAC to ignore all serial port errors, in which case the data bytes are sent even if they contain serial port errors.

NSER 0

The GPIB-MAC reports this error when the GPIB-MAC detected no serial port error as a result of the last operation.

EPAR 1

The GPIB-MAC records this error when the parity of the received character is not what was expected. This means

that 1 or more bits of the received character were corrupted in such a way as to change the character's parity.

EORN 2

The GPIB-MAC records this error when characters arrive at the serial port faster than the serial port can accept them When this error occurs, one or more characters sent to the serial port have been lost. If this error occurs, check to see that the GPIB-MAC and your Macintosh are using the same serial port settings.

EOFL 3

The GPIB-MAC records this error when the GPIB-MAC's internal serial port buffer overflows. This should only occur if XON/XOFF is disabled.

EFRM 4

The GPIB-MAC records this error when a character is received whose stop bits are not in the expected place. This can happen when the number-of-bits-per-character setting of the GPIB-MAC does not match your Macintosh. It can also happen if the baud rates of the GPIB-MAC and your Macintosh do not match, or if one side of the serial link does not use parity and the other side does.

Appendix C
Changing from 115 Volts AC to 230 Volts AC

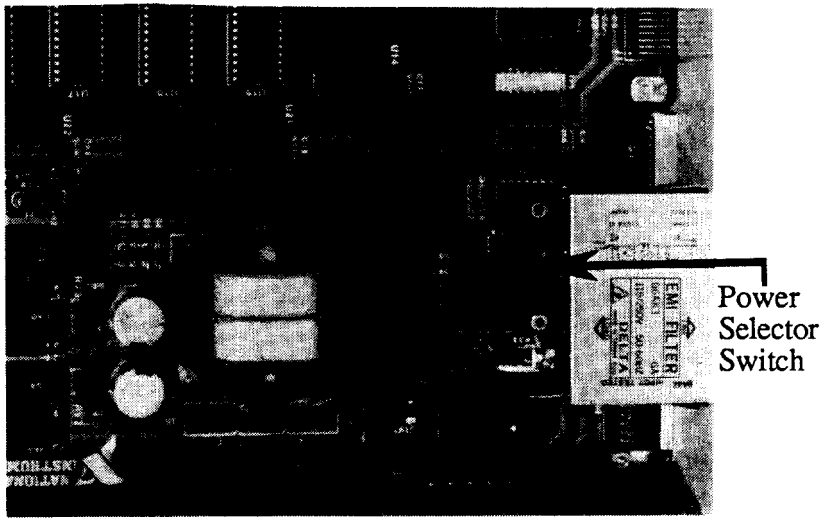
This section shows you how to convert your GPIB-MAC from 115 Volts AC to 230 Volts AC.

1. Turn the power switch to Off. This switch is located on the rear panel of the GPIB-MAC.

2. Disconnect the power cord from the power source and from the rear panel of the GPIB-MAC.

3. Remove the cover from the GPIB-MAC by first removing the two screws on both sides of the housing. Lift off the cover.

4. Change the setting of the red power selector switch so that it reads 230V. The following figure shows the location of the power selector switch.



GPIB-MAC with Cover Removed

5. Replace the cover. Be sure the aluminum side plates are in the proper place. Replace the screws you removed in Step 1.

6. Remove the fuse assembly. Replace the 1/4 ampere fuse supplied with the GPIB-MAC with a 1/8 ampere fuse. Re-insert the fuse assembly.

Appendix D
Operation of the GPIB

The GPIB is a link, or bus, or interface system through which interconnected electronic devices communicate. Hewlett-Packard invented the GPIB, which they call the HP-IB, to connect and control programmable instruments manufactured by them. Because of its high system data rate ceilings of from 250K bytes to 1M byte per second, the GPIB quickly became popular in other applications such as intercomputer communication and peripheral control. It was later accepted as the industry standard IEEE-488. The versatility of the system prompted the name General Purpose Interface Bus.

Types of Messages

Devices on the GPIB communicate by passing messages through the interface system. There are two types of messages:

* Device-dependent messages, often called data or data messages, contain device-specific information such as programming instructions, measurement results, machine status, and data files.

* Interface messages manage the bus itself. They are usually called commands or command messages. Interface messages perform such functions as initializing the bus, addressing and unaddressing devices, and setting devices for remote or local programming.

  The term command as used here should not be confused with some device instructions which are also call commands. Such device-specific instructions are actually data messages.

Talkers, Listeners, and Controllers

There are three types of GPIB communicators. A Talker sends data messages to one or more Listeners. The Controller manages the flow of information on the GPIB by sending commands to all devices.

Devices can be Talkers, Listeners, and/or Controllers. A digital multimeter, for example, is a Talker and may also be a Listener. A printer or plotter is usually only a Listener. A computer on the GPIB will often combine all three roles to manage the bus and communicate with other devices.

The GPIB is a bus like a typical computer bus except that the computer has its circuit cards interconnected via a backplane bus whereas the GPIB has standalone devices interconnected via a cable bus.

The role of the GPIB Controller can also be compared to the role of the computer's CPU, but a better analogy is to the switching center of a city telephone system.

The switching center (Controller) monitors the communications network (GPIB). When the center (Controller) notices that a party (device) wants to make a call (send a data message), it connects the caller (Talker) to the receiver (Listener).

The Controller usually addresses a Talker and a Listener before the Talker can send its message to the Listener. After the message is transmitted, the Controller usually unaddresses both devices.

Some bus configurations do not require a Controller. For example, one device may only be a Talker (called a Talk-only device) and there may be one or more Listen-only devices.

A Controller is necessary when the active or addressed Talker or Listener must be changed The Controller function is usually handled by a computer.

System Controller and Active Controller

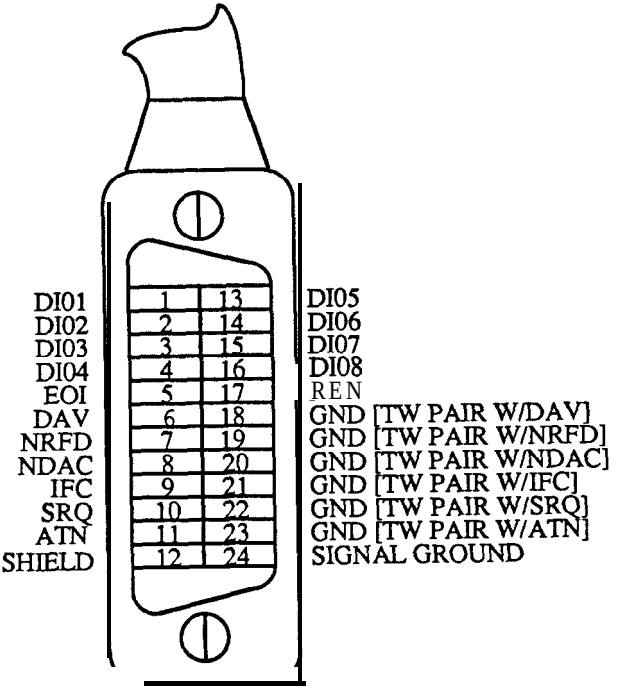
Although there can be multiple Controllers on the GPIB, only one Controller at a time is Active Controller or Controller-in-Charge (CIC). Active control can be passed from the current Active Controller to an idle Controller. Only one device on the bus, the System Controller, can make itself the Active Controller.

GPIB Signals

The interface bus consists of 16 signal lines and 8 ground return or shield drain lines. The 16 signal lines are divided into three groups:

* 8 datalines
* 3 handshake lines
* 5 interface management lines

The following figure shows the arrangement of these signals on the GPIB cable connector.



GPIB Cable Connector

Data Lines

The eight data lines, DIO1 through D108, carry both data and command messages. All commands and most data use the 7-bit ASCII or IS0 code set, in which case the 8th bit, D108, is unused or used for parity.

Appendix A lists the GPIB command messages.

Handshake Lines

Three lines asynchronously control the transfer of message bytes among devices. The process is called a three-wire interlocked handshake and it guarantees that message bytes on the data lines are sent and received without transmission error.

NRFD (not ready for data)

NRFD indicates when a device is ready or not ready to receive a message byte. The line is driven by all devices when receiving commands and by Listeners when receiving data messages.

NDAC (not data accepted)

NDAC indicates when a device has or has not accepted a message byte. The line is driven by all devices when receiving commands and by Listeners when receiving data messages.

DAV (data valid)

DAV tells when the signals on the data lines are stable (valid) and can be accepted safely by devices. The Controller drives DAV when sending commands and the Talker drives it when sending data messages.

The way in which NRFD and NDAC are used by the receiving device is called the Acceptor Handshake. Likewise, the sending device uses DAV in the Source Handshake.

Interface Management Lines

Five lines are used to manage the flow of information across the interface.

ATN (attention)

The Controller drives ATN true when it uses the data lines to send commands and false when it allows a Talker to send data messages.

IFC (interface clear)

The System Controller drives the IFC line to initialize the bus to become Controller-In-Charge.

REN (remote enable)

The System Controller drives the REN line, which is used to place devices in remote or local program mode.

SRQ (service request)

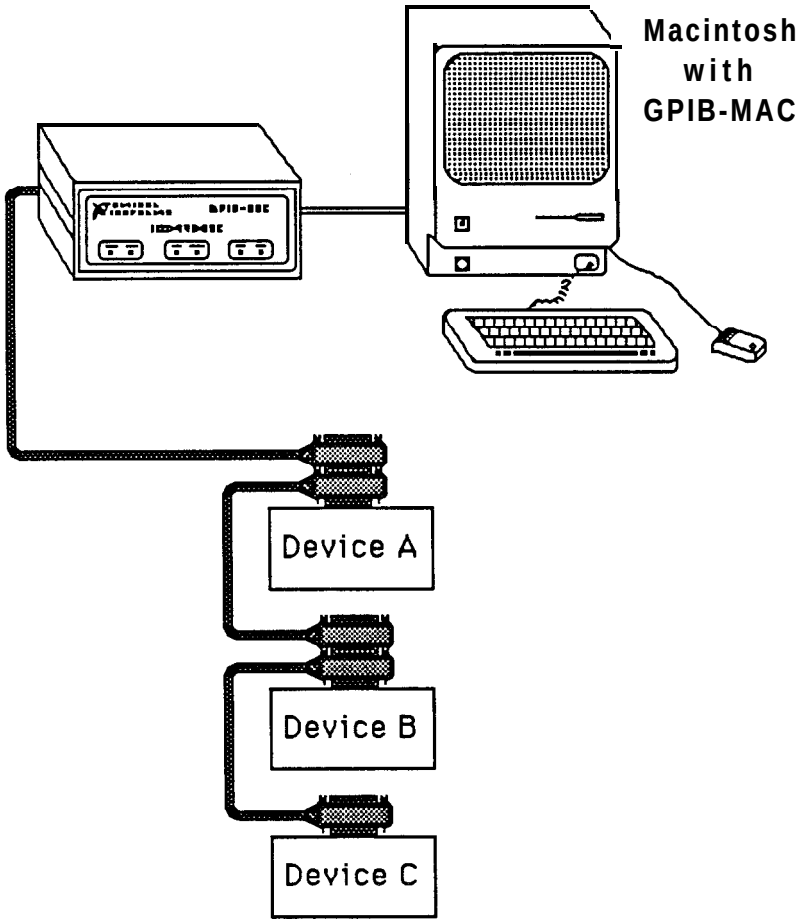
Any device can drive the SRQ line to asynchronously request service from the Active Controller with the SRQ line.

EOI (end or identify)

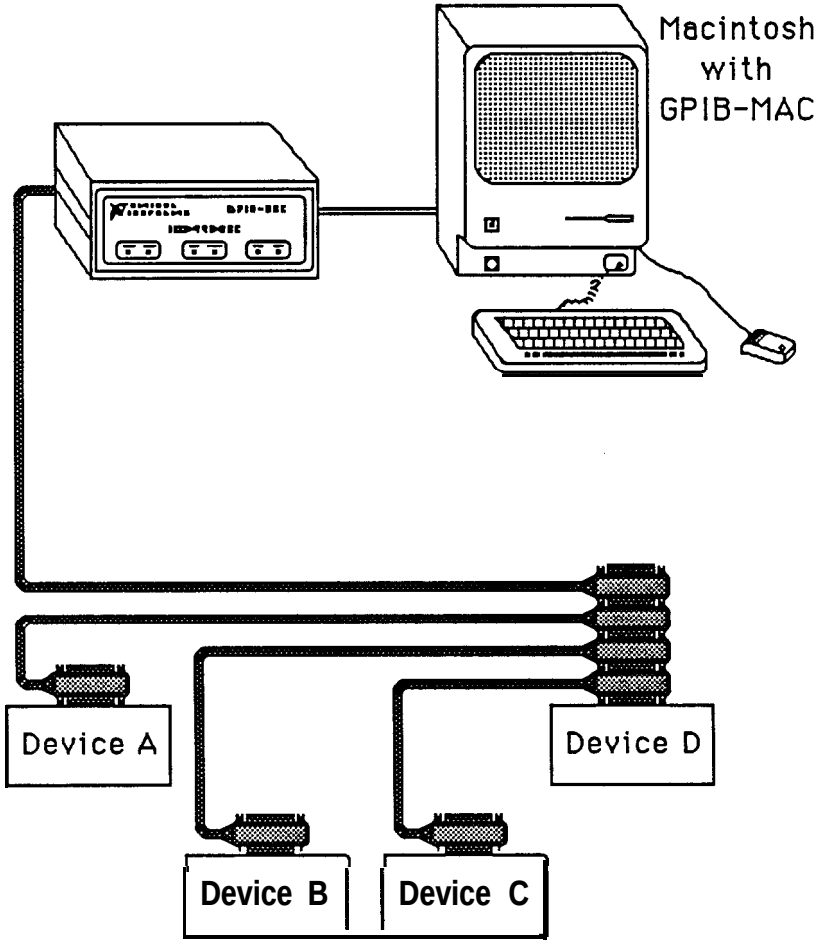
The EOI line has two purposes. The Talker uses it to mark the end of a message string. The Active Controller uses it to tell devices to identify their responses in a parallel poll.

Physical and Electrical Characteristics

Devices are usually connected with a cable assembly consisting of a shielded 24-conductor cable with both a plug and receptacle at each end. This design allows devices to be connected in either a linear or a star configuration, or a combination of the two. See the following figures.



Linear Configuration of GPIB Devices



Star Configuration of GPIB Devices

The standard connector is the Amphenol or Cinch Series 57 MICRORIBBON or AMP CHAMP type. An adapter cable using non-standard cable and/or connector is used for special interconnect applications.

The GPIB uses negative logic with standard TTL logic levels. When DAV is true, for example, it is a TTL low level (<= 0.8V), and when DAV is false, it is a TTL high level (>= 2.0V).

Configuration Restrictions

To achieve the high data transfer rate that the GPIB is designed for, the physical distance between devices and the number of devices on the bus is limited.

The following restrictions are typical:

* A maximum separation of four meters between any two devices and an average separation of two meters over the entire bus.

* A maximum total cable length of 20 meters.

* No more than 15 devices connected to each bus, with at least two-thirds powered on.

Bus extenders are available from National Instruments and other manufacturers for use when these limits must be exceeded

Appendix E
Common Questions

This appendix gives answers to common questions.

Question

Why does the manual suggest that I use INPUT$ sometimes and LINE INPUT# at other times? Microsoft suggests using INPUT$ to read from the serial port

Answer

Use LINE INPUT# to read status information from the GPIB-MAC. GPIB-MAC software formats its status information so that your BASIC program may easily read and interpret each of its pieces. Each logical piece of status information is followed by a carriage return and linefeed. LINE INPUT# allows you to easily read each piece of status information and assign it to a variable.

Use INPUT$ to read a data string from your GPIB-device. INPUT$ requires that you know the exact number of characters you wish to read from the serial port. When reading status information from the GPIB-MAC, this is not always possible since the responses may vary in length from one call to the next. But when reading a data string from your GPIB device, you requested a certain number of bytes and you should use INPUT$ to read the number of bytes you requested in your **rd** function. The GPIB-MAC appends to the end of the data string a string containing the number of bytes that were actually read from the GPIB. So once you read in the data bytes using INPUT$, use LINE INPUT# to read the string containing the byte count. Refer to the example following the **rd** function description in Section Four.

Question

When I use LINE INPUT# my strings are usually preceded by a linefeed and followed by a carriage return. Why don't my strings contain both a carriage return and linefeed at the end?

Answer

LINE INPUT# stops reading when a carriage return is seen and does not skip over the linefeed in the sequence. The linefeed is not read until the following LINE INPUT#. In most cases you will be using the VAL function to convert the string to a value and a leading linefeed is ignored.

Question

I sent the programming message "rsp 10" to the GPIB-MAC to serial poll device 10. Then, I used LINE INPUT# to read the response byte and got nothing but a carriage return and linefeed as a response. Am I doing something wrong?

Answer

No. To conduct a serial poll, the GPIB-MAC must be Controller-In-Charge or it must be able to become Controller-In-Charge. If the GPIB-MAC cannot become Controller-In-Charge no serial poll is conducted and, therefore, you will not get a response string. To see if this is the problem, ask for status (see **stat)** and check to see if the ECIC error occurred. If it did, you have passed control or system controller authority to some other GPIB device, and will not be able to perform a serial poll until the GPIB-MAC gets controller authority back.

Appendix F
Parallel Polling

A Parallel Poll allows a GPIB controller to obtain information from several devices on the GPIB in one operation. The controller polls configured devices and reads back a single response byte that contains one bit of information from each device. From this information the controller can determine which devices need servicing.

Operation

A device is configured by sending it its listen address and a parallel poll enable (PPE) message. There are 16 possible PPE messages, hex 60 - hex 6F. The bits in the PPE message have the following meaning:

		U	S	DIO lines 1-8
0	11	X	X	X X X

U - when 0 (hex 6X), parallel poll is enabled. when 1 (hex 7X), parallel poll is disabled

S - when the device's ist (individual status) bit matches the S bit, the device will assert the appropriate data line true. hex 60 - hex 67 set S to 0, hex 68 - hex 6F set S to 1.

DIO The value "n" in bits O-2 corresponds to one of the DIO lines l-8, where n corresponds to DIO line n+l. Thus a value of 2 (binary 010) corresponds to DIO line 3.

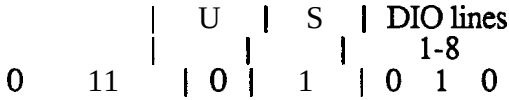
The circumstances under which a device sets its ist bit are specific to that device. For example, a device might always set the ist bit to 1 when it is busy and 0 when it is free, or vice versa. (Consult your device documentation for this information.) With this information the controller can configure devices according to the information desired.

Only the active controller can perform a parallel poll.

There are two steps to conducting a parallel poll: the configuration step and the actual polling. The following paragraphs describe these two steps.

Configuration

The ppc function configures devices for parallel polls. For example, if you want to configure a device at address 5 to respond on DIO line 3 when the ist bit is 1, the programming message would be **ppc 5,3,1**. The GPIB-MAC takes the arguments 3,1 and constructs the following parallel poll enable byte:



The value of this byte is hex 6A where

U = 0, enable
S = 1, thus when ist = 1 the device will assert DIO line3 (which corresponds to 010 in bits O-2).

The ppc function sends the device's listen address, the Parallel Poll Configure command, hex 6A, then the Unlisten command. The active controller can configure itself to respond to a parallel poll using ppc, also. This might be used in the case where the GPIB-MAC is not the system controller and the system controller does not have the capability to do the configuration. Since the GPIB-MAC cannot be the controller, the GPIB controller in your system must configure the GPIB-MAC in order to parallel poll it.

The Parallel Poll

After configuring the device, the GPIB-MAC now conducts a parallel poll by calling rpp. In the previous example, where the device was sent a configuration byte of hex 6A, if the device's ist bit matches the S bit of hex 6A, rpp will return the value 04. Here, the third least significant bit is set, corresponding to DIO line 3. (If any other devices responded positively on other lines, those corresponding bits would be set as well.)

Note that the controller may configure more than one device to respond on the same data line, in which case the bits in the response byte are set by the ORing of all the responses on that line.

Disabling Parallel Poll Response

The active controller may disable a specific device from responding to a parallel poll by calling ppu with the device address as a parameter. ppu sends the device the Parallel Poll Disable command hex 70 (binary 01 **1 1 0000),** which sets U to 1 to disable the device from responding to a parallel poll.

To unconfigure all devices, the controller may call ppu with no arguments, which sends PPU (parallel poll unconfigure, hex 15).

Example

A system has three line printers, two tape drives, one card reader, and one PC on a system. The PC uses a GPIB-MAC to communicate on the GPIB. All other devices are GPIB devices. The PC is designated to be active controller, and all other devices recognize this. Furthermore, all devices will set their ist bit to 1 when they are busy and 0 when they are free.

The active controller configures the card reader (at address 6) to respond positively on DIO line 4 (which sets bit 3 of the response byte) when free by sending the configuration byte 01100011. (The S bit is set to 0, the value of bits O-2 is 3.)

PPC 6,4,0

When a parallel poll is conducted one of two things will happen. If the card reader is free, bit 3 of the response byte will be 1; if it is busy, bit 3 will be 0. When the device is free its ist bit is 0 and because this equals the value of the S bit, the device asserts DIO line 4.

The active controller configures the line printers to all respond positively on DIO line 1 when busy. In this case, the ppr,s argument for each of them is 0,1. Thus, the configuration byte for each of them is 01101000 (hex 68). When a parallel poll is conducted, the controller can immediately find out if all line printers are free because the response in this situation will be 0. If any line printer is busy, bit 0 of the parallel poll response will be 1, corresponding to DIO line 1 being asserted. But what if the active controller wants to know if one line printer is free? If the controller reconfigures the line printers to respond positively when free (dio,ss = 0,0;0 configuration byte = 01 1000000), then if any device is free, it will drive the DIO line to 1. Thus,the controller can use S-bit/ist bit correspondence for different types of information.

Appendix G
Setting Switches

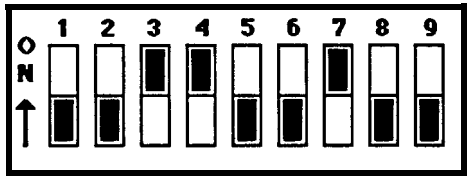
This appendix explains how to set the switches on the back panel of the GPIB-MAC.

You may change the serial port characteristics of the Macintosh from within BASIC or other programming languages.

The default characteristics of the Macintosh modem port are as follows:

baud rate: 9600
parity: none
data bits: 8
stop bits: 2

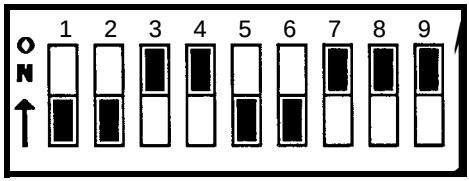
If these defaults meet your needs, set the switches on the GPIB-MAC as follows:



You may want to change the default characteristics of the serial port on the Macintosh. For instance, you may want to run at a higher baud rate (up to 57.6K baud). To change the default characteristics to 57.6K baud, no parity, 8 data bits, and 2 stop bits from within Microsoft BASIC, place the following BASIC statement at the beginning of your program:

```
OPEN "COM1:57600,N,8,2" AS #1
```

then set the switches on the GPIB-MAC as follows:



Remember, whatever serial port characteristics you decide to use, you must set up both your Macintosh and your GPIB-MAC to the exact same characteristics.

Appendix H
Sample Program

This appendix contains general programming steps and a sample program. These are meant to be guides for you as you start writing your programs for the GPIB-MAC.

General Steps

The steps below are general programming steps. The following pages contain explanations of these steps and show a sample program.

Step 1 Send **stat** function to have status information returned to you after your programming message.

step 2 Send serial port initialization functions if you need to change default serial port settings.

step 3 Send GPIB initialization functions if you need to change default GPIB settings.

step 4 Communicate with the device using the **rd** and **wrt** functions, and check status if you requested it.

After you initialize the GPIB-MAC, the **rd** and **wrt** functions may be the only functions you will need.

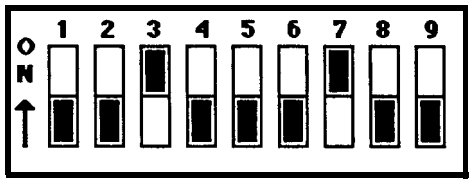
Using an HP 7475A Plotter with a Macintosh

This example shows how to write a program on the Macintosh using Microsoft BASIC to draw a circle using an HP 7475A Plotter.

Getting Ready to Program

Before you start programming, determine the serial port settings you will use. The settings for this example are: 9600 baud rate, 8 data

bits, 1 stop bit. and no parity. Set the back panel switches on the GPIB-MAC as follow;



Then, connect the serial cable to the serial port of your Macintosh and to the GPIB-MAC. Connect the GPIB cable to your device and the GPIB-MAC. Turn the power switch of the GPIB-MAC to On.

Programming Steps

Step 1 - stat Function

In BASIC, before reading or writing to the serial port, a device must be "opened" Place the following BASIC statement at the beginning of your program to open and configure the serial port (COM1) and name it device #1.

```
OPEN "COM1:9600,n,8,1" as #1
```

Note that you will now use PRINT #1 to redirect strings to the serial port. Now, send the **stat** function if you want status information returned after every programming message. To do this, include the following code in your program:

```
PRINT # 1, "stat c n"
```

After you send this programming message, you can expect four lines of data at the serial port (each line is terminated by <CR><LF>). For now, call a subroutine to check the status. You can write this later. For now, add the line:

```
GOSUB status
```

Step 2 - Serial Port Functions

Send serial port initialization programming messages, if necessary. You can skip this for this example.

Step 3 - GPIB Initialization Functions

Send GPIB initialization programming messages, if necessary. Again, you can skip this for this example.

Step 4 - Communicate with rd and wrt Functions

Communicate with the device using **wrt** programming messages, and reading back status after each. Here is the heart of the program! After each **wrt** string, call the subroutine status which will check for errors. The plotter's GPIB address is 5.

```
PRINT #1,"wrt 5"
PRINT #1,"in;sp1,pa1000,3000;ci500;"
      GOSUB status

status:
STAT%=VAL(LINE INPUT #1, status$)
LINE INPUT #l,gpiberr$
LINE INPUT # 1 ,sperr$
LINE INPUT # 1 ,cnt$
PRINT status!§ gpiberr$ sperr$ cnt$
if stat% < 0 GOSUB error
```

Following is the completed program.

```
OPEN "com1:9600,n,8,1" AS #1
PRINT #1,"stat c n"
GOSUB status
PRINT # 1 ,"wrt 5"
PRINT #1,"in;sp1,pa1000,3000;ci500;"
      GOSUB status
END
```

```
status:
STAT%=VAL(LINE INPUT #1, status$)
LINE INPUT # 1 ,gpiberr$
LINE INPUT # 1 ,sperr$
LINE INPUT # 1 ,cnt$
PRINT status$ gpiberr$ sperr$ cnt$
if stat% < 0 GOSUB error

error
REM Place your code to handle errors here.
STOP
```

Appendix I
Serial Port Sample Program

This appendix contains a sample program that contains code to control the serial port of the Macintosh from a C program. It is provided to demonstrate how to write your own serial I/O routines if the language you are using does not already provide them..

Sample Program

```
/* send.c g-30-85 JR.  Send file out modem port, fast.
 * :bk=0;
 * :ts=2;
 * Purpose:  This program is designed to send a text file
 * containing commands and data to the GPIB-MAC.
 * The program was compiled with Manx Aztec C-68K 106.F.
 • The data file is read into memory once, then sent to the
 ▫ serial driver in a loop.
 * Programming languages such as MacBasic and MacPascal
 * have too much overhead to send data faster than 19200
 * baud.
 */

#include <quickdraw.h>
#include <pb.h>       /* File & Device managers */
#include <stdio.h>    /* Standard I/O */
#include <serial.h>
#include <memory.h>
#include <event.h>
#include <inits.h>    /* InitFonts */
#include <packages.h> /* File Package ● /
#include <textEdit.h> /* textbox */

#define FALSE 0

main0
{
    short refAOut, refAIn;    /* port reference numbers ● /
    char ● pBufIn;            /* input buffer */
    int rc;
/* result code */
    int inLen = (1<<13);      /* input buffer size */
```

```
    InitGraf( &thePort);
    InitFonts();                /* initialize Fonts */
    /* call OS Event Mngr to discard events */
    FlushEvents( everyEvent,0);
    InitWindows();
    TEInit();                   /* Initialize TextEdit *
    InitDialogs( NULL);
    InitCursor();
    /* Clear the Screen */
    EraseRect( &thePort->portRect);
    TextFont(4);                /* Monaco */
    TextFace(0);                /* plain */
    TextSize(9);

    /* open serial drivers */
    if( rc = OpenDriver( "\P.AOut", &refAOut))
        myExit( "Can't open modem port, rc= %d.\n", rc);
    if( rc = OpenDriver( "\P.AIn", &refAIn))
        myExit( "Can't open modem port, rc= %d.\n", rc);

    /* set baud, stop, databits */
    setupport( refAIn, refAOut);

    /* make buffer too big for input to overrun ● /
    for( inLen=(1<<14); inLen>32 && pBufIn==NULL; inLen>>1)
        /* must be room here somewhere ● /
        pBufIn = NewPtr( (long)inLen);
    if( SerSetBuf( refAIn, pBufIn, inLen))
        tbox("SerSetBuf error.");

    /* load file and send */
    fload( refAIn, refAOut);
}/* main */


setupport( refIn, refOut)
short refIn, refOut;

    short parity, stopbits, databits, baud:
    char c;
    SerShk flgs;                /* serial port handshake flags */
    int rc;                     /* result code */
    EventRecord theEvent;

    /* set serial port attributes */
    flgs.fXOn = TRUE; /* output flow control */
    flgs.fCTS = FALSE;          /* not hardware flow control */
    flgs.xOn  = '\021';         /* ^Q ● /
```

```
    flgs.xOff = '\023';       /* ^S */
    flgs.errs = hwOverrunErr | framingErr;
    flgs.evts = 0;
    flgs.fInX = FALSE;        /* not input flow control */
    if( rc = SerHShake( refOut, &flgs))
        myExit("SerHShake error, rc= %d\n", rc);
    baud = baud57600;
    puts("Current baud = 57600, enter 's' for 19200.");
    while( !GetNextEvent( keyDownMask, &theEvent))
        ;                     /* wait for key down */
    if( (char)theEvent.message =='s')
        baud = baud19200;
    parity = noParity;
    stopbits = stop20;
    databits = data8;
    SerReset(refIn,(short) (baud | parity | stopbits| databits));
    SerReset(refOut, (short) (baud | parity | stopbits |databits));



myExit( str, rc)
char *str;
int rc;

    printf( str. rc);
    exit(0);



fload( refIn, refOut)
/* load and send a file */
short refIn, refOut;

    char *s;
    SFReply reply;
    StringHandle hStr;
    short refNum;
    int rc;            /* result code */
    register long pause;
    long logEOF;       /* length of file */

static SFTypeList typeList = {'TEXT');
static Point   where = { 100,70};

    SFGetFile(pass(where),"", OL, 1, typeList, OL. &reply);
    while (reply.good) {
        EraseRect( &thePort->portRect);
        if( SetVol( NULL, reply.vRefNum)){
```

```
            tbox( "Can not SetVol") ;
            continue;

        if ( FSOpen( reply.fName, reply.vRefNum, &refNum)){
            tbox("Can't open file");
            continue;

        /* get end of file ● /
        if (rc= GetEOF( refNum, &logEOF)){
            tbox("Can't find EOF");
            continue:

        printf("EOF = %ld.\n", logEOF);
        /* get buffer for file ● /
        s = NewPtr( logEOF);
        if (s== NULL){
            tbox("File size too large for memory"):
            continue:

        /* read file into buffer */
        if ( FSRead( refNum, &logEOF, s)){
            tbox("Could not read file.");
            continue;

        printf("Bytes read into buffer = %ld\n", logEOF);
        FSClose( refNum); /* don't need input file now ● /
        send( s, logEOF, refOut);
        DisposPtr( s);        /* release buffer ● /
        puts("Waiting for any response from device.");
        for( pause=TickCount()+30; TickCount() < pause;)

        showIn( refIn);
        SFGetFile(pass(where),"", OL, 1, typeList, OL, &reply);



send( s, logEOF, refOut )
char *s;
long logEOF;
short refOut;

    unsigned short loops = 1;

    puts("# loops: ");
    scanf("%hd", &loops);
    printf("--Total bytes to send= %ld bytes--\n", logEOF*loops);
    while ( loops--)(
        FSWrite( refOut, &logEOF,s);       /* send file */
```

```
showIn( refin)
short refin;

    long count, pause;
    char *text:

    /* Read any bytes waiting at the modem port */
    SerGetBuf(refin, &count);        /* get count */
    if (count > 0){
        text = NewPtr( count):       /* get another buffer */
        if ( text == NULL){
               tbox("Port input size too large for memory");
               return:

        printf("Number of bytes at input port= %ld.\n", count);
        if( FSRead( refin, &count, text)) {
               tbox("Can't read input port buffer");
               return;

        /* write text to screen */
        write( 1, text, (int)count);
        for( pause=TickCount()+60; TickCount()< pause;)

    }else
    tbox("No bytes at input port.");


tbox( txt)
char *txt;

    Rect r;
    register long pause:

    SetPort( thePort);
    r = thePort->portRect;
    OffsetRect(&r, -r.left, -r.top);
    InsetRect(&r, (r.bottom - r.top) >> 2, (r.right - r.left) >>
2);
    TextBox(txt, (long)strlen(txt), hr. teJustCenter);
    FrameRect(&r);
    for( pause=TickCount()+60; TickCount() < pause;)
        ;
```

Product User Comment Form

National Instruments encourages you to give us your comments on the clarity and accuracy of the documentation supplied with its products. This information helps us continue providing high quality products to meet your needs.

Did you find deficiencies? Please comment on the completeness, clarity, and organization.

_____

_____

_____

Did you find errors in the manual? If so, please give the page number, and a description of the error.

_____

_____

_____

Thank you for your help.

Name_____ Title _____

Company Name_____

Number & Street _____

City_____ State & Zip_____

Phone (___) _____

Mail to: Technical Publications
National Instruments
12 109 Technology Blvd.
Austin, TX 78727

GPIB-MAC